École doctorale de Saint-Étienne

Extraction de données et apprentissage automatique pour les sites web adaptatifs

Thèse présentée à l'école nationale supérieure des mines de Saint-Étienne
pour obtenir le grade de :

Docteur de l'école nationale supérieure des mines de Saint-Étienne et

de l'université Jean Monnet de Saint-Étienne

Mention : Informatique.

par Thierry Murgue

Rim
Centre G2I

Eurise
Faculté des Sciences et Techniques

Soutenue le 12 décembre 2006, devant le jury composé de :

Thierry Artières . . . . . Maître de conférences, Lip6, Paris
Jean-Jacques Girardot . . . Maître de recherche, Ensmse, Saint-Étienne, Co-directeur
Colin de la Higuera . . . . . Professeur, Ujm, Saint-Étienne, Co-directeur
Philippe Jaillon . . . . . Ingénieur de recherche, Ensmse, Saint-Étienne, Encadrant
Robert Mahl . . . . . Professeur, Ensmp, Paris, Rapporteur
Jean-Marc Petit . . . . . Professeur, Insa, Lyon, Président du jury

À Stéphanie pour son soutien, sa compréhension, son amour ;
À Robin : que ses sourires continuent d'embellir ma vie.

Remerciements

Sur la page de garde, seul figure mon nom comme auteur de ce manuscrit. Même si, par définition on attribue le travail de thèse à une seule personne : il n'en est rien. Personnellement, j'ai eu la possibilité de travailler durant tout mon doctorat avec une multitude de personnes, je vais les remercier pour leur soutien et leur aide précieuse qui m'ont permis de mettre en œuvre tout ce que je décris dans ce manuscrit. D'autres personnes ont eu un impact décisif sur ce travail, mais de manière moins directe : ce sont les membres de ma famille et mes amis, je les remercierai aussi.

Tout d'abord, j'aimerai remercier Jean-Marc Petit, professeur à l'Insa de Lyon, d'avoir accepté de présider mon jury de thèse.

Plus particulièrement, je tiens à remercier Robert Malh, professeur à l'École des Mines de Paris, d'une part, Patrick Gallinari et Thierry Artières respectivement professeur et maître de conférence au laboratoire d'informatique de Paris 6, d'autre part, pour avoir accepté de rapporter ce travail de thèse. La qualité de leurs rapports m'a permis de corriger grandement ce mémoire.

Je tiens à dire un grand merci à Jean-Jacques Girardot, Colin De La Higuera et Philippe Jaillon pour avoir su proposer un sujet en lien avec des activités de recherche communes de l'équipe Rim du centre G2I (qui s'appelait encore Simmo lorsque j'ai commencé) d'une part, et de l'Eurise d'autre part. J'ai ainsi pu bénéficier d'une expérience enrichissante en effectuant ces travaux de recherche au sein de deux entités : l'École des Mines et l'université Jean Monnet.

Ces dernières années, je les ai vécues au sein de l'équipe Eurise, dont je voudrais remercier de nouveau son directeur : Colin. J'aimerais lui exprimer toute ma gratitude aussi bien pour son encadrement de qualité, son accueil dans l'équipe de recherche et surtout pour tout ce qu'il a pu m'apporter au niveau humain. Son encadrement ne s'est pas arrêté à la thèse, mais il a été (et est encore) un véritable tuteur : il m'a permis de grandir scientifiquement. Je lui fais part de mon plus sincère respect. L'équipe Eurise ne serait pas ce qu'elle est sans cette complicité instaurée par les membres et la bonne humeur que ces derniers y font régner. À Abdallah, Baptiste, Catherine, Christine, Codrin, David, Fabrice, Franck, François, Jean, Jean-Christophe, les « Marcs », Mathias et Philippe, Merci !

Pour finaliser le manuscrit, François Jacquenet et Fabrice Muhlenbach l'ont relu et ont corrigé le fond et la forme à maintes reprises. Merci à eux ! Dans la continuité, je voudrais remercier Jacques André, qui ne me connaît pas, que je ne connais pas, mais qui a eu la bonne idée d'écrire un recueil intitulé « Petites leçons de typographie » [And90].

J'en arrive maintenant à vouloir rendre hommage à mes chers collègues étudiants : je pense évidemment en premier à ceux que j'ai eu l'honneur de suivre pendant toutes mes études. Donc pour Amaury, Cyrille, Steven, Maxime. Merci ! Mes nouveaux collègues et doctorants de l'EURISE sont aussi présents dans mes pensées, je remercie pour leur sincère camaraderie Stéphanie, Henri-Maxime, Toufik, Rémi et Frédéric.

Même si je les ai déjà remerciées un peu plus haut en tant que collègues, je fait part de mes pensées amicales à ces deux personnes.

Amaury est la personne avec qui j'ai fait toutes mes études supérieures. Nous avons eu de sacrés bons moments, et je tiens à le remercier tout particulièrement pour toutes les réponses qu'il a volontiers apporté à mes questions parfois — souvent — grotesques. Je le remercie aussi pour sa bonne humeur constante lors de « virées » à Chalmazel, Grenoble ou encore Montpellier, de matchs auxquels nous avons assistés ensemble à Geoffroy Guichard, de parties de pétanque ou de fléchettes acharnées.

Dans le même esprit, j'ai une pensée toute particulière pour Marc Bernard qui, entre autres qualités, dessine super bien une maison, un jardin, un établi, une scie circulaire. Un grand merci à lui, pour son aide : je n'oublie pas sa relecture de certains papiers et son entière disponibilité, sa bonne humeur, son goût des choses bien faites, nos discussions éclectiques en salle café : du vin à l'éducation des enfants en passant par l'esthétisme (ou la possibilité d'afficher sur un serveur X des fenêtres plus ou moins transparentes `:-)`).

Une immense pensée à ma bande de potes : Mig et Mag (et Bastien), Mike et Marie, Kiki et Roro, **L'**Éric et **La** Mag (et leurs puces), Juju et Jean-Marc (et Matthieu), Boubou et Lio. Merci d'être là !

Je voudrais maintenant remercier mes proches. Et en premier, mon épouse. Mariée à un « courant-d'air », Stéphanie m'a apporté un soutien sans faille. Elle a été présente à chaque fois où le *blues* du doctorant, tel un spectre, faisait une discrète apparition. Pendant certaines périodes de travail, j'ai répondu négligemment « oui, oui » à la quasi totalité de ses questions, sans pour autant être capable de reformuler la question quelques secondes après. Je lui demande de m'excuser pour ces moments de non-présence à ses côtés.

À mes parents, je voudrais leur faire part de ma gratitude pour m'avoir poussé à « bien travailler à l'école ». Grâce à eux, j'ai eu de bonnes conditions pour gravir les premiers échelons des études supérieures. Mais non, « je n'ai pas continué après maths sup ! ». Un petit coucou à Jéjé, Val et ma filleule Jade ; ainsi qu'à Joëlle, Francis, Aurélie

et Ludo.

Un remerciement à blake, jerry, speedy, averell, jack, obelix, thorgal, corto, mickey, lapinot, eurise, depinfo, cian, lanfeust, hebus, joe, fantasio, lagaffe, mortimer, nemo, mortadelo, idefix, asterix, nicolede, spirou, hobbes, calvin, mafalda, filemon, william, garfield, luckyluke, rahan, tom, blueberry, cafe, tisane, chocolat, numerobis, amonbofis, epidemais, cixi, titeuf, panoramix, petitsuix, abraracourcix, agecanonix, assurancetourix, bonemine, cetautomatix, falbala, ielosubmarine, ordralfabetix, plantaquatix ; le sous-réseau `161.3.6.0/24` puis `161.3.203.0/24` ; debian `woody` puis `sarge` (et `etch` !) ; mon palm.

Je voudrais clore cette liste en remerciant ici les personnes que j'aurai pu oublier...

--
Thierry

Sommaire

Avant-Propos

Les travaux que nous présentons dans ce mémoire se positionnent dans le paradigme d'extraction de connaissance à partir de données. Le contexte web des données utilisées nous fait nous placer dans le cadre du *Web Usage Mining*. Chacune des parties de ce manuscrit possède une page de présentation résumant les contributions développées dans la partie en question. Ces contributions portent sur le prétraitement des données web et l'évaluation des modèles de langages en inférence grammaticale stochastique.

Dans le sommaire, les parties, chapitres et sections dont l'intitulé est suivi du caractère $\star$ reflète une vision des problèmes qui nous est propre et/ou marquent des contributions importantes de la thèse.

Introduction ⋆

1 Présentation du problème ⋆

De nos jours, le *Web* est un lieu incontournable d'échange d'informations. De simples ensembles de pages web statiques à sa naissance, celui-ci a évolué ces dernières années vers la mise à disposition de services de plus en plus complexes. Suivant le type de site développé — commerce électronique, bibliothèque numérique, journal d'actualités, forum de discussions, etc. — les services peuvent être variés : de l'achat d'un bien à la rencontre de l'âme sœur, en passant par la lecture de son journal préféré en ligne. C'est sous l'impulsion des acteurs du commerce électronique, cherchant à fidéliser leurs clients, qu'ont été développées des recherches visant à fournir aux usagers la meilleure qualité de service possible. Dans cette perspective, il parait naturel d'adapter à la personne navigant sur leur site, le service auquel elle accède. Ce constat va maintenant bien au delà de la philosophie commerciale : en effet, la plupart des sites désirant améliorer les services proposés se tournent vers le concept de sites web adaptatifs [PE97].

Ce besoin d'adaptation en fonction de l'utilisateur induit une méthodologie de développement de sites web centrée sur celui-ci. En considérant cette approche, les concepteurs ont alors besoin pour améliorer la qualité du site en fonction d'un utilisateur, de connaître les informations le concernant. En fait, une grande quantité de données peut être récupérée suite à la navigation d'un utilisateur sur un site. La taille sans cesse croissante de celles-ci et leur diversité ne permet toutefois pas à l'être humain de traiter de manière manuelle cette information. Une manière classique d'aborder alors le problème consiste à suivre le paradigme d'*extraction de connaissance à partir de données* (ECD). Défini dans [FPSM91], l'ECD est un processus complet de traitement de l'information de manière semi-automatique dans le but d'extraire de la connaissance. Il se décompose en plusieurs étapes illustrées sur la figure 1.

Les trois premières étapes de l'ECD, la *sélection*, le *nettoyage* et le *formattage* constituent le prétraitement des données. Dans le cas original du paradigme présenté précédemment, les données d'entrées sont des bases de données structurés. Certains travaux montrent qu'elles renferment souvent des données incohérentes, manquantes ou erronées qu'il faut donc traiter avant la phase d'apprentissage. Dès le milieu des années 90, des chercheurs s'intéressent à des données moins structurées issues du *Web*. Lorsque le processus est appliqué à ces données (pages web, *logs*[1], liens entre pages, ...), l'étape

[1] De la terminologie anglophone : enregistrements des communications bilatérales de demande (côté

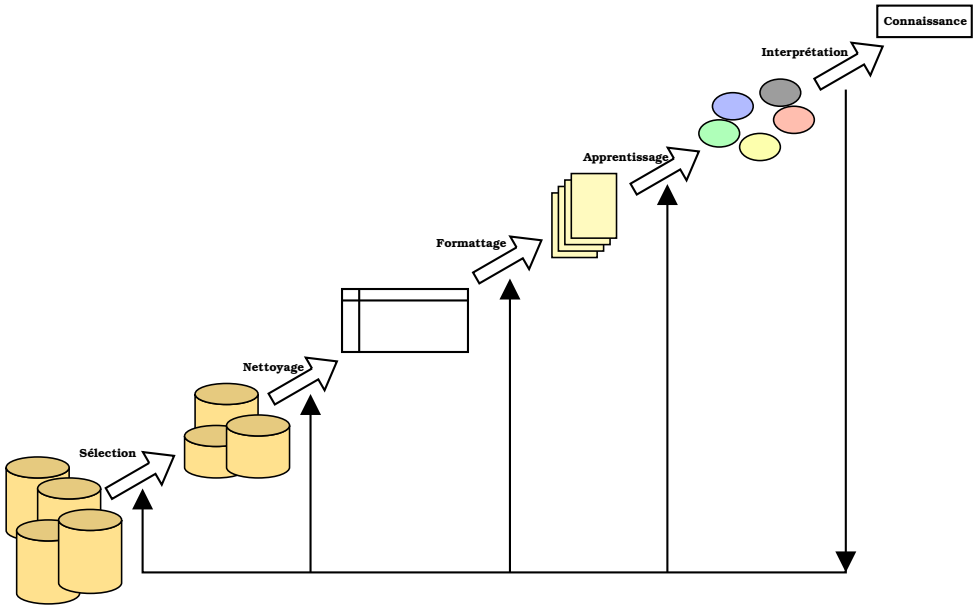


FIG. 1 – *Le processus d'extraction de connaissance à partir de données*

de prétraitement est encore plus délicate. Les pages web ne partagent pas de structuration commune, ce qui rend la tâche de traitement automatique difficile. De nombreux travaux traitent du prétraitement des données dans le cadre du *Web*. L'article [PPR96] traite de la structure ou topologie du site pour améliorer les données d'entrée. D'autres travaux [PP99, FMK02] traitent de l'utilisation de modèles appris en prédiction. Les articles [CMS99] et [PPPS01] présentent respectivement un système complet pour l'ECD dans le contexte du *Web* et un inventaire des techniques existantes dans le contexte de *Web Mining*.

L'activité réseau enregistrée par les serveurs ne correspond pas toujours à l'activité des utilisateurs. En effet, une dégradation des données existe lorsque les échanges passent par des serveurs proxy ou cache. Les problèmes généraux liés aux données de navigation sont décrits dans [CPY96, Pit97, SBKF01, MDLN01, CBC03]. Certains travaux [Pit97, CMS99] présentent les dégradations constatées et donnent des pistes pour reconstruire les données telles qu'elles devraient être s'il n'y avait pas eu ces machines. Nous irons plus loin en évaluant une méthode de reconstruction des données initialement dégradées.

L'étape suivante du processus d'ECD est appelée *Data Mining* ou fouille de données. Suivant le contexte, elle est aussi appelée *apprentissage automatique* ou *apprentissage artificiel*. De nombreux travaux ont été menés sur cette étape dans le cadre de bases de données structurées, et depuis la deuxième moitié des années 90 des chercheurs ont adaptés les techniques utilisées alors, dans le cadre du *Web*. Dès [CSM97], Co-

client) et d'envoi (côté serveur) de l'information.

OLEY *et al.* présentent le terme de *Web Mining* comme l'union de deux sous domaines traitant de données de nature différente. Le *Web Content Mining* s'intéresse au traitement des données constituées du contenu de pages web. Le *Web Usage Mining*, quant à lui, s'intéresse aux informations concernant l'utilisation (ou les utilisateurs) des sites. Plus récemment, le *Web Structure Mining* traite [KB00] le cas où la structure du site web est incluse dans les données d'entrée.

Le domaine de l'apprentissage automatique vise à développer des méthodes, techniques et outils visant à améliorer, au cours du temps, les performances de la machine en construisant, à partir de données (exemples positifs et/ou négatifs), un modèle plus général que ces données d'entrée. Par exemple, en considérant les caractéristiques associées à une rencontre de football entre deux équipes (nombre de jours de repos avant la rencontre, classement respectif de l'équipe dans le championnat, température du terrain, etc.), nous définissons comme exemple les ensembles de caractéristiques ayant mené à une victoire de la première équipe, et comme contre-exemples les autres. Les techniques développées en apprentissage automatique pourraient construire un modèle qui prédirait (avec une certaine qualité), en fonction de nouvelles caractéristiques, la victoire de telle ou telle équipe dans une rencontre ultérieure. Considérons un autre exemple, cette fois, issu de la reconnaissance de parole. D'un ensemble de sons et de leur retranscription en langue naturelle, un modèle appris pourrait, depuis une séquence sonore qui n'appartenait pas à l'ensemble des sons de départ, donner sa retranscription. Le fait que le modèle appris puisse traiter une information qu'il n'a encore jamais rencontrée provient de ce que celui-ci a généralisé les données d'entrée. En considérant le cas de données représentant au mieux les navigations des utilisateurs de sites web, on peut envisager d'utiliser des techniques d'apprentissage automatique afin de construire, par l'observation des comportements des utilisateurs, un modèle général de ceux-ci. Ainsi muni de ce modèle, il nous sera possible d'adapter un site web en fonction d'une navigation en cours. C'est dans ce contexte spécifique du vaste domaine qu'est le *Web Usage Mining* que se situent les travaux développés dans cette thèse. Nous utilisons aussi la structure du site web pour prétraiter les données et, en ce sens, nous nous situons aussi dans le contexte du *Web Structure Mining*.

Dans le contexte général du processus d'ECD où les données d'entrées sont des bases de données structurées, des méthodes à bases de règles d'associations [AS94] et d'arbres de décisions [Qui93] sont souvent utilisées. Mais, dans le domaine que nous étudions, une caractéristique intrinsèque est l'absence de contre-exemples (appelés aussi *données négatives*) : en effet, en considérant les navigations des utilisateurs, il est difficile de spécifier qu'un comportement n'est pas un « bon » comportement. Cela nous conduit alors à nous intéresser à l'apprentissage de modèles stochastiques bien adaptés à ce genre de situation. Il existe une autre contrainte forte, liée au domaine : l'obtention de modèles appris intelligibles pour que les concepteurs de sites web puissent les analyser et en tirer des règles pour, soit adapter le site web manuellement, soit définir une interface d'adaptation automatique en fonction des résultats du modèle. Ces contraintes guident notre choix de méthode d'apprentissage vers l'*inférence grammaticale stochastique* [CO94a, RST95, LPP98, Tho00, CT04, HO04]. En effet, cette dernière est connue pour inférer des modèles structurels simples à analyser (les *automates*), à partir de don-

nées séquentielles positives seulement. Il est à noter que peu de travaux [KPPS94, BL99] utilisent des modèles grammaticaux dans le contexte du *Web Usage Mining*.

Une des problématiques liées à l'apprentissage automatique est l'évaluation des modèles appris. En effet, comment savoir si un modèle est bon ? Comment savoir s'il correspond à ce qu'il nous faut chercher ? Une réponse possible consiste à mesurer l'écart entre le modèle appris et ce qu'il est censé représenter. La divergence de Kullback-Leibler entre deux modèles stochastiques est une grandeur souvent utilisée dans la littérature qui représente l'écart en terme de codage informatique (binaire) entre les représentations des deux modèles. Cette mesure, bien qu'étant largement utilisée dans l'état de l'art [ABCF94, KD02, HTV+05, HBS05]), souffre de deux défauts importants : ce n'est pas une distance au sens mathématique et elle peut être infinie. Ainsi, il nous est impossible d'effectuer des opérations topologiques sur les modèles appris qui pourraient permettre d'accélérer les algorithmes d'inférence grammaticale ou même de comparer les écarts entre-eux. Les cas où cette mesure est infinie ne sont pas forcément les cas où les deux modèles sont très différents en terme de reconnaissance ou de prédiction, ce qui est problématique pour l'évaluation des modèles appris. De plus, lorsque la mesure est infinie, le recours à des techniques de lissage est obligatoire et introduit un biais dans l'évaluation des modèles. Nous proposons donc une distance entre modèle de langages qui ne souffre pas de ces maux.

2 Organisation de la thèse ⋆

Ce mémoire de thèse se compose de trois parties. Une première partie traite de l'acquisition des données du *Web* (étapes 1 à 3 du processus d'ECD). Une deuxième s'intéresse à l'inférence grammaticale stochastique (étape 4 de ce même processus) et enfin une dernière présente des expérimentations permettant de valider les travaux des deux précédentes.

Dans la première partie, nous présentons tout d'abord les différentes définitions des concepts liés au réseau, au *Web*, aux données issues de ce dernier. Une étude la plus exhaustive possible est menée dans le chapitre 2 concernant toutes les données disponibles en lien avec l'utilisation des sites web par les usagers. Nous justifions ensuite notre choix de travailler sur les enregistrements des serveurs (*logs*), et nous consacrons un chapitre à leur description. Dans ce dernier, nous présentons les dégradations constatées sur ces *logs* et les méthodes que nous développons pour rendre aux données leur fiabilité ainsi perdue. Finalement, nous montrons comment passer de ces transactions enregistrées à des séquences — format de données requis en entrée des algorithmes d'inférence grammaticale utilisés.

La deuxième partie est consacrée à l'inférence grammaticale stochastique. Elle représente une instanciation possible de la quatrième étape du processus d'extraction de connaissance à partir de données. Nous donnons dans le chapitre 4 les définitions classiques des objets employés et rappelons quelques propriétés de ceux-ci. Le chapitre 5

décrit formellement l'apprentissage automatique, c'est à dire définit ce que veut dire « bien apprendre » en fonction des données d'entrée, de cadres formels, des modèles utilisés. Nous insistons sur deux cadres d'apprentissage largement décrits dans la littérature : l'identification à la limite et le cadre PAC. Nous consacrons un chapitre complet à la présentation des algorithmes d'apprentissage par fusions d'états, ceux-ci étant à la base de la majorité des recherches menées en inférence grammaticale. Nous nous intéressons ensuite aux modèles appris et surtout à leur qualité. Ainsi, le chapitre 7 présente la perplexité, mesure utilisée pour l'estimation de la qualité des modèles appris, et quelques méthodes de lissage. Ce dernier permet d'obtenir une perplexité finie pour tout modèle. Le lissage introduit un biais dans l'évaluation du modèle car c'est le couple modèle-lissage qui est pris en compte, plutôt que le modèle seul. Enfin dans le dernier chapitre de cette partie, nous donnons les définitions de notre adaptation de la distance d_2 aux modèles de langages. Puis, nous montrons les résultats théoriques et pratiques que nous avons obtenus sur l'utilisation de cette dernière. Nous désirons faire prendre conscience qu'il peut être important d'utiliser une vraie distance pour comparer les modèles, sans avoir de recours obligé au lissage de ces derniers.

La dernière partie valide expérimentalement les méthodes développées dans les deux parties précédentes. Sur des données artificielles, dans un premier temps, nous montrons que les modèles appris sur des données prétraitées sont plus proches des cibles prévues. Ensuite, sur des données réelles, nous utilisons nos modèles pour prédire la prochaine page de la navigation d'un utilisateur en connaissant les pages déjà analysées. Cette tâche est classique dans le domaine du *Web Usage Mining*. De plus, nous avons la possibilité, en utilisant un protocole identique, de nous comparer aux résultats obtenus par GERY *et al.* [GH03] sur cette tâche.

Enfin dans la conclusion, nous résumons les différentes méthodes développées pour résoudre le problème initialement posé et donnons quelques perspectives pour continuer ce travail.

Première partie

Extraction des données web

Les travaux présentés dans cette partie ont donné lieu à une communication en conférence nationale et à une version améliorée dans un livre :

[Mur05a] T. Murgue, *De l'importance du pré-traitement des données pour l'utilisation de l'inférence grammaticale en Web Usage Mining*, Atelier sur la modélisation utilisateurs et personnalisation de l'interaction homme-machine (EGC'05), 2005.

[Mur06] ———, *Extraction Des Connaissances : État Et Perspectives*, RNTI, vol. E5, ch. Modélisation d'utilisateurs et Personnalisation de l'Interaction Homme-Machine, Cépaduès, Toulouse, France, 2006, ISBN :2.85428.707.X.

Ils ont aussi été présentés lors d'une conférence internationale :

[MJ05] T. Murgue et P. Jaillon, *Data Preparation and Structural Models for Web Usage Mining*, Sciences of Electronic, Technologies of Information and Telecommunications (SETIT'05) (Sousse, Tunisie), 2005.

It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.
Sir A. Conan Doyle [Doy91]

1 Définitions

Sommaire

Résumé

Nous proposons dans ce travail d'extraire des informations à partir de données provenant du *Web*. Nous présentons dans ce chapitre quelques définitions et notations pour guider le lecteur tout au long de ce manuscrit. Dans l'ordre, nous donnons notamment quelques informations sur *internet*, une de ses applications : le *Web*, les concepts réseaux associés. Nous proposons des éléments de discussion au sujet d'une éthique du *Web*.

1.1 Introduction ⋆

La recherche d'information, la consultation de données et l'achat en ligne, sont des exemples de l'utilisation du *Web*. Dans le but d'améliorer ces services — et d'autres — nous nous intéressons aux informations liées au comportement des utilisateurs de cet environnement. L'immersion dans le monde de l'internet implique la connaissance d'un vocabulaire et de concepts dédiés. Nous présentons les réseaux de communication employés sous différentes formes, de l'entité conceptuelle au réseau physique. C'est dans cet espace que nous pouvons récupérer une quantité d'informations pertinentes pour comprendre comment les utilisateurs se servent des outils dans le but d'améliorer les services qui leur sont destinés. Pour cela, nous avons besoin de données disponibles sous des formes variées : c'est le point de départ de tout le processus visant à établir un modèle de comportement défini en lien avec la navigation d'une personne.

Pour fixer les idées, nous introduisons dans ce chapitre les diverses notions que nous utiliserons tout au long de ce document.

1.2 Internet

Le terme « Internet[1] » définit le concept de réseau d'inter-connexions, ou plus simplement de « réseau de réseaux ». Il existe dans le monde de nombreux réseaux d'interconnexions, pour cela les anglophones font une distinction entre « internet » et Internet (avec un `i` majuscule) qui représentent respectivement soit n'importe lequel des réseaux d'inter-connexions, soit le plus grand. Pour les français, la différence se fait à un autre niveau ; on emploie généralement le terme internet sans article défini pour référencer le réseau physique et avec un article, l'internet, pour définir le concept du réseau global. Dans le langage courant, la distinction n'est pas faite : on parle de manière indifférenciée d'internet ou de l'internet. Dans le reste de notre manuscrit, nous emploierons internet comme référence au réseau physique d'inter-connexions le plus grand du monde.

Internet, donc, est historiquement issu d'un petit réseau de machines financé par le gouvernement américain : quelques machines communiquent entre elles ; ARPAnet[2] est né ! Le but expérimental de ce réseau était de pouvoir communiquer entre deux machines par temps de guerre avec des parties physiques de réseaux détruites. Les protocoles initiaux employés pour satisfaire cette contrainte sont `Tcp/Ip` : ces derniers gèrent une structure physique de réseau avec des chemins redondants pour faire transiter de l'information entre deux nœuds définis, ainsi en cas de dysfonctionnement de quelques nœuds, la quasi-totalité des communications continue de circuler. L'émergence d'autres réseaux américains a conduit à la mise en place d'un réseau global : internet. Cet espace d'échanges devenu mondial autorise divers types de communications : messages électroniques, jeux en ligne, téléchargement de fichiers.

Le *Web*, ou de manière plus précise le *World Wide Web* (littéralement la « toile mondiale », généralement abrégé en www ou w3), ne représente qu'une application *hypertexte* répartie sur internet.

1.3 Présentation du *Web*

Le terme « hypertexte » date des années soixante : il définit une structure de document non linéaire. La possibilité de lire ce type de document a mené à la création du *Web*. Avant de poursuivre, nous invitons le lecteur à découvrir quelques concepts de base.

Définition 1.1 (Page web) *Nous appelons « page web » un ensemble de ressources (texte, image, vidéo, son, ...), accessibles par une seule adresse sur le réseau (*Uri[3]*)*

1 Le mot *Internet* provient de l'anglais *Interconnected Network*

2 ARPAnet est le sigle correspondant à *Advanced Research Projects Agency Network*

3 De l'anglais *Uniform Resource Identifier*

Définition 1.2 (Site web) *Un site web est constitué d'un ensemble de pages web interconnectées.*

> *Remarque :* Le World Wide Web Consortium (w3c), instance qui normalise et développe les modèles et outils pour l'utilisation du *Web*, définit un site web comme l'ensemble des pages interconnectées d'un même lieu sur le réseau, accessibles depuis une page « hôte ».

Le *Web* est une application distribuée dans plusieurs endroits réseaux, abritant les sites web. L'architecture essentielle du *Web* est non symétrique : des *serveurs* diffusent l'information à des *clients* en ayant fait la demande. Les termes « serveur » et « client » sont ambigus : serveur se réfère aussi bien à la machine matérielle qu'à l'outil logiciel qu'elle héberge et qui sert les pages ; client se réfère aussi bien à la machine (voire même l'utilisateur de cette dernière) qu'au navigateur qui demande l'information et permet de la visualiser.

Le *Web*, depuis sa création, n'a cessé d'évoluer en terme de taille : en 1990, au CERN à Genève, le premier site web voit le jour et comporte une unique page ; le moteur de recherche Google [Goo] comptait plus de huit milliards de pages en septembre 2005 ; aujourd'hui il ne publie plus le nombre de pages indexées. La grande popularité du *Web* peut s'expliquer en partie par l'adéquation parfaite entre protocoles réseaux et communication développés et les besoins d'un tel système : outre les protocoles TCP/IP qui gèrent l'acheminement de l'information sur le réseau, le protocole HTTP, de niveau d'abstraction plus élevé, est utilisé pour la navigation au sein des documents hypertextes.

1.4 Protocole HTTP

Ce protocole[4] a été développé dans le but de servir les documents hypertextes sur le *Web*. Il a été normalisé [W3C99b], dans sa version actuelle, par le W3C. La partie qui nous intéresse est très simple et peut être décrite ainsi : un client demande une ressource, le serveur lui répond. La figure 1.1 montre un exemple de communication

FIG. 1.1 – *Le très simple protocole* HTTP

directe entre client et serveur *via* le protocole HTTP. Dans un premier temps, le client

[4] HTTP est l'acronyme de *HyperText Transfer Protocol*

demande au serveur des informations concernant la ressource se trouvant à l'Uri `http://www.site.fr/A`. Le serveur, à la réception de la requête, teste la disponibilité de la ressource demandée et envoie une réponse (dans l'exemple « OK »).

La normalisation du protocole définit trois différents types de requêtes. Chacun de ces types de requêtes (ou *méthodes*) devraient[5] être utilisés pour des buts précis, la réponse du serveur n'étant pas identique. La méthode GET est la plus utilisée, elle correspond à une demande de contenu identifié par une Uri, le serveur — si la ressource est disponible — envoie une réponse constituée d'informations sur la ressource demandée, suivies de la ressource elle-même. HEAD représente une requête similaire à GET, à la différence près que la réponse du serveur ne contient pas la ressource demandée mais simplement des informations sur celle-ci. La dernière méthode POST est formalisée dans le but de joindre à la requête une entité destinée à modifier l'Uri concernée. Par exemple, poster un message sur un forum de discussions, annoter une ressource existante, envoyer les données d'un formulaire ou ajouter des données dans une base de données *via* le *Web* sont typiquement des actions devant recourir à la méthode POST.

Les réponses envoyées par le serveur sont également normalisées. Elles sont composées d'un code de retour, d'informations sur le serveur, d'informations sur la ressource concernée suivis, le cas échéant, de la ressource ou du message d'erreur. Les codes sont des nombres de trois chiffres et sont répartis en famille : ceux s'écrivant `1xx` (`x` représentant un chiffre quelconque) informent, ceux en `2xx` donne une réponse positive à la requête, `3xx` pour ceux qui renvoient sur une autre ressource, `4xx` pour une réponse négative à la requête du client, `5xx` lorsqu'une erreur est survenue sur le serveur durant le traitement de la requête.

Ce protocole permet ainsi d'envoyer et de recevoir de l'information contenue dans les documents hypertextes. Ces derniers, pour être lus par les navigateurs clients, doivent suivre une syntaxe précise définie en une famille de langages dits « à balises » : Sgml et ses langages dérivés tels que Html ou Xml.

1.5 Famille Sgml

La majeure partie des documents présents sur le *Web* sont des documents écrits en Html. Sgml [ISO86] est une standardisation permettant de définir la structure d'un document et la relation entre ses parties, Html [W3C92] n'est qu'un format de document issu de Sgml. Le langage Html a connu un véritable engouement lors de la naissance du *Web*, il facilitait la création de document hypertexte en définissant la structure du document, l'apparence des éléments le composant, et des liens pour naviguer entre les documents, ou des parties de ceux-ci. Un document Html peut ainsi contenir des éléments de différents types : du texte, des images, du son, de la vidéo, *etc.*

La grande complexité de Sgml, la mauvaise structuration de Html (mélangeant structure et présentation), ont poussé le W3C à imaginer un langage successeur pour le *Web*. En 1998, Xml [W3C98] est ainsi apparu comme un sous-ensemble de Sgml

[5] mais ce n'est pas toujour le cas

simple et permettant la définition rapide de documents. Dans un souci de cohérence et

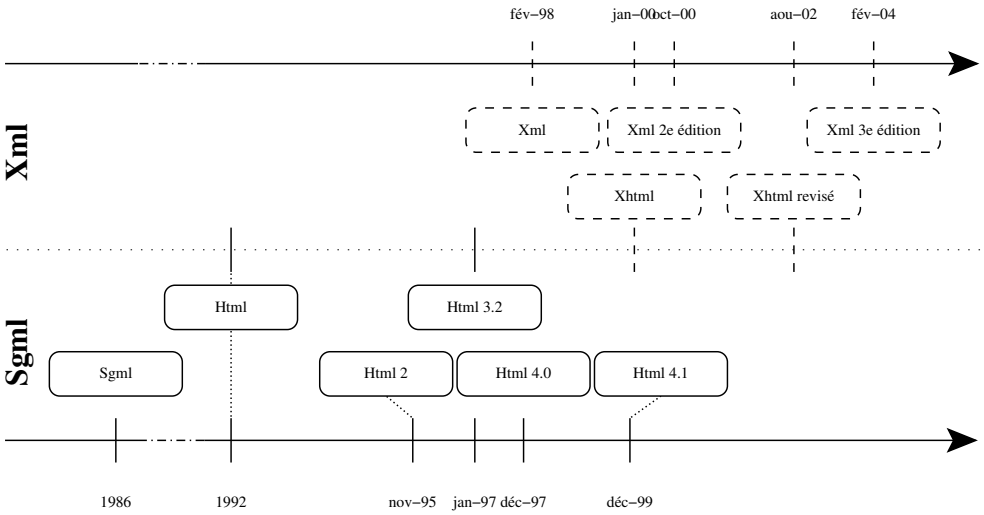


FIG. 1.2 – *Évolution de la famille* SGML

pour profiter de l'assise du langage HTML dans le monde du *Web*, le W3C a décidé de faire évoluer le langage du *Web* vers XHTML [W3C00]. Ce dernier est la réécriture de la dernière version[6] de HTML [W3C99a] dans le format simplifié XML.

Une pages web est le plus souvent associée à un document HTML. Nous avons vu comment, grâce au protocole HTTP, ces pages étaient envoyées à partir du serveur et reçues par le client. La modélisation idéale qui fait correspondre une demande de document à une seule requête au serveur puis à une seule réponse de ce dernier ne représente que rarement la réalité : en effet, nous allons voir en quoi certaines évolutions peuvent compliquer les échanges.

1.6 Fonctionnalités additionnelles

L'échange simpliste de la figure 1.1 ressemble dans la réalité souvent à celui décrit dans la figure 1.3. Les diverses fonctionnalités ajoutées sont décrites ci-après.

Définition 1.3 (Cache [Pit97]) *La* mise en cache *est définie comme un « mécanisme visant à restreindre le temps de récupération d'une ressource en sauvegardant une copie de celle-ci localement ».*

Ainsi, lors de la requête d'un document, si une copie de ce dernier est disponible dans le cache, elle est renvoyée comme réponse sans que le serveur correspondant réponde

6 Cette version sépare la structure du document de sa représentation *via* le concept de feuilles de styles

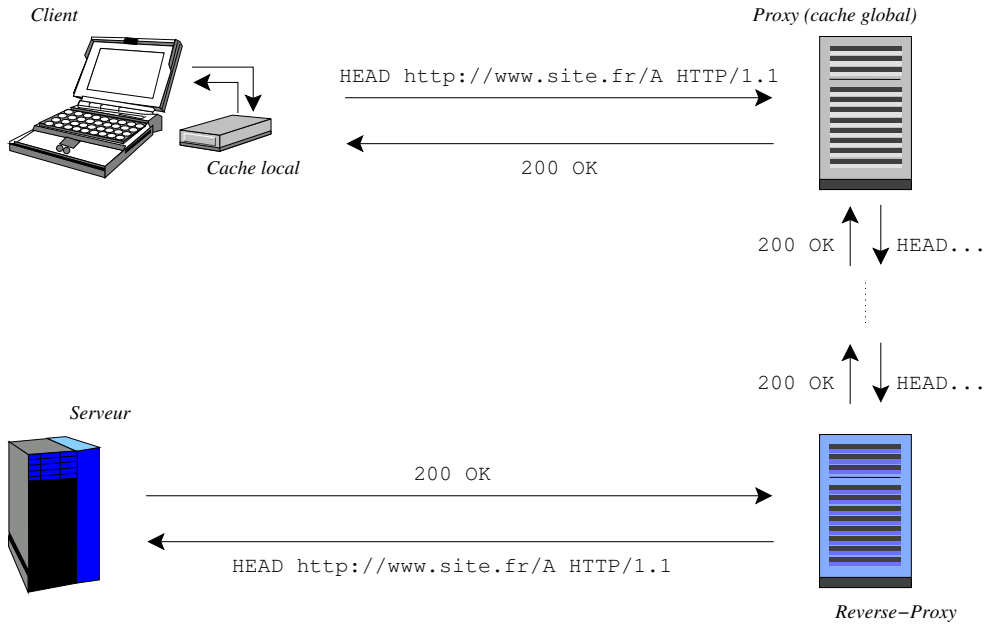


Fig. 1.3 – *Le protocole* Http *avec fonctionnalités additionnelles*

à une requête, sinon la requête classique est envoyée. Les algorithmes employés et les limitations du cache classique sont développés dans [ASA+95].

Définition 1.4 (Proxy) *Le terme anglophone proxy fait référence à une machine intermédiaire placée quelque part entre le client et le serveur, qui permet d'acheminer indirectement les requêtes de l'un vers l'autre.*

La présence de ce type de matériel dans l'architecture réseau dédiée au *Web* peut s'expliquer suivant deux points de vue : sécurité et performance. Tout d'abord, ce type de machines est généralement placé en sortie d'un réseau d'une grande entité (campus, grande entreprise, ...), elles permettent de gérer plus finement la sécurité en n'autorisant les connexions sur le *Web* que depuis cette unique machine : les requêtes de tous les clients de l'entité sont ainsi relayées par le *proxy*.

De plus, associé à un système de mise en cache global — par comparaison au cache dit « local » souvent présent sur le navigateur —, ce type de machine permet de diminuer le trafic réseau entrant en mettant en cache une importante quantité de données : plus les personnes de l'entité hébergeant le *proxy* émettent des requêtes identiques (lecture de documentation, accès aux mêmes données), plus le gain de la mise en cache global est important.

Définition 1.5 (Reverse proxy) *Le reverse proxy comporte les mêmes fonctionnalités que le proxy vu du côté serveur. Dans la pratique le fonctionnement est similaire.*

Si le *proxy* permet une plus grande souplesse dans la gestion de la sécurité pour l'accès à internet des clients, le *reverse proxy* fait de même pour les serveurs : par exemple,

dans une entité disposant de plusieurs serveurs web, un *reverse proxy* peut servir à n'autoriser les connexions depuis le *Web* que sur cette unique machine, les requêtes vers les autres serveurs étant relayées. De même, associé à un système de mise en cache global, il permet la réduction du trafic sortant.

Les points de suspension entre le *proxy* et le *reverse proxy* de la figure 1.3, nous indiquent qu'il n'existe pas de limite sur le nombre d'intermédiaires entre le client et le serveur.

Les versions 1.0 et 1.1 du protocole HTTP définissent des directives de gestions des caches pendant l'acheminement de l'information. La première recommandation du W3C mentionne une directive PRAGMA de type générique pouvant accepter des couples (clef, valeur) modifiant l'acheminement des données. Elle accepte aussi l'unique mot-clef NO-CACHE : dans ce mode, le client doit toujours recevoir une réponse du serveur et non d'un intermédiaire. Dans sa deuxième version, le protocole d'échange est muni de la directive CACHE-CONTROL permettant une plus grande souplesse pour la définition des types de caches (sur disque, en mémoire, ...), la date d'expiration des données, *etc*.

Ces fonctionnalités sont communément utilisées pour une plus grande souplesse de navigation et sont la source de requêtes n'aboutissant pas au serveur (le projet akamai [Aka] a pour but d'accélérer la totalité d'internet par des méthodes de cache). Le *prefetching* qui consiste à télécharger les données en tâche de fond pour qu'elles soient disponibles dans le cache lorsque l'utilisateur en aura besoin, entraîne lui aussi une différence entre la navigation d'un utilisateur et la façon dont sont vues les requêtes côté serveur.

Nous verrons dans la suite du document que lorsque nous nous plaçons du côté du serveur pour suivre la navigation d'un client, ces perturbations dans la gestion des requêtes peuvent être gênantes. Des solutions consistant à passer outre ce type d'architecture existent mais elles vont à l'encontre des principes qui ont conduit à la mise en place des fonctionnalités additionnelles. Ce que nous devons ou pouvons faire pour améliorer le service mis à disposition de l'utilisateur est discuté ci-après.

1.7 Éthique et déontologie du *Web* ⋆

Dans les années 1990, avec l'augmentation de l'utilisation d'internet (plus précisément des newsgroups), de nombreuses personnes se sont posées les questions d'éthique sur le *Web*. Une recommandation de l'IETF[7] [IET95] définit quelques règles sur la disponibilité des données sur un site web, tandis qu'un livre plus générique explique [VS94] les « bonnes manières » sur internet. Des règles de bonne conduite sont définies dans ce dernier, notamment concernant le respect de la vie privée et la sauvegarde de la bande passante. Mais qu'en est-il au niveau de la législation sur ces deux points ?

7 Acronyme pour Internet Engineering Task Force

1.7.1 Respect de la vie privée ⋆

De manière internationale, divers textes régissent les problèmes de respect ou non de la vie privée. Jusqu'aux années 2000, la plupart des textes ne faisaient aucune référence à toute la partie électronique de la vie privée. Les textes dont nous disposons dans le droit français datent de plusieurs années et viennent de tous horizons. Tout d'abord, le code civil napoléonien [MBT+04] définit la vie privée dans son article 9. Depuis, en 1948, la déclaration universelle des droits de l'homme [ONU48] définit les « immixtions arbitraires dans [la] vie privée ». Plus récemment, la convention européenne des droits de l'homme reprend la notion de vie privée. Enfin, seule la « loi relative à l'informatique, aux fichiers et aux libertés » [ÉF78] de 1978, définit le traitement électronique des données nominatives. Il est à noter qu'une adresse IP peut être considérée comme « indirectement nominative ».

Le code pénal et les jurisprudences française et européenne sont une mine d'informations sur les traitements informatiques, mais aucun texte n'a encore réussi à dégager un consensus international sur les divers points concernant la vie privée.

Dans ce qui nous importe, les données gardées sont constituées de pages demandées à un instant défini, par une machine spécifique. L'identification, même indirecte, de l'utilisateur n'est pas évidente. Lorsque l'objet même de cette partie est d'avoir des traces d'utilisateurs fiables, il faut garder à l'esprit les limites de tels procédés.

1.7.2 Ressources réseaux ⋆

Les problèmes liés à la vie privée ont vu leur nombre exploser ces dernières années. À l'encontre de ce phénomène, les soucis d'utilisation non « responsable » des ressources réseaux et notamment de la bande passante n'ont pas aboutis à une législation claire. En France, il n'existe pas de loi, définissant le bon usage du réseau physique, par contre les article $323-1$ et suivants du nouveau code pénal [ÉF92], définissent les sanctions encourues lors d'une attaque en vue de « fausser ou entraver le fonctionnement d'un système de traitement automatisé de données ». L'appréciation humaine doit donc être mise en avant pour spécifier si une surcharge réseau peut entrer dans ce cas là. En effet, la mise en place d'un système gourmand en ressources réseau pourrait paralyser ce dernier, et donc être répréhensible.

Ainsi, en considérant les quelques paragraphes des codes spécifiant ces abus, ou tout simplement en prenant en compte les « bonnes manières » de l'internet, il existe un cadre plus ou moins légal pour ne pas utiliser à l'excès les ressources réseaux, ce qui provoquerait une gêne sensible pour d'autres services.

1.8 Conclusion ⋆

Ce premier chapitre présente le contexte dans lequel se déroule notre travail. L'objectif principal du processus que nous décrivons est d'obtenir des informations sur les utilisateurs de sites web. Nous avons présenté les concepts centraux de réseau, de *Web*

ainsi que la famille des protocoles utilisés. Nous avons également vu dans quelle proportion il existe des cadres et une certaine déontologie dans l'utilisation des données du *Web*.

Le chapitre suivant traite des données elles-mêmes : que sont-elles, de quelle manière les récupérer ? Ces deux questions, pour être traitées de manière adéquate, demandent une grande connaissance des données disponibles. Une étude la plus exhaustive possible des informations pouvant être récupérées est primordiale. C'est ce que nous poposons de faire dans le chapitre suivant où nous analysons les données disponibles du côté client et du côté serveur.

2 Données du Web et leur acquisition ⋆

Sommaire

Résumé

Nous rappelons que le but de notre travail est de fournir à l'utilisateur d'un site web une meilleure qualité de services. Pour y parvenir, nous voulons connaître les différentes manières qu'ont les utilisateurs de naviguer afin d'en déduire la personnalisation adéquate pour le service demandé. Nous présentons dans ce chapitre les divers types de données liées au comportement des utilisateurs et leur accessibilité. Nous argumentons notre choix de données pour notre problème.

2.1 Introduction ⋆

Le processus complet défini dans notre travail relève comme nous l'avons vu dans l'introduction (section 1), de l'extraction de connaissance à partie de données. Dans ce paradigme, l'étape de sélection, souvent négligée, définit complètement l'espace de travail pour le reste du processus.

Les données en rapport avec la navigation sur le *Web* se repartissent en deux grands groupes : celles disponibles lorsque nous nous plaçons du côté client du protocole `HTTP`, et celles du côté serveur. En effet, suivre la navigation d'un utilisateur et avoir accès à tout ce qui se passe sur son poste de travail rend disponible une grande quantité de données, toutes dépendantes de l'environnement de l'utilisateur en question. À l'inverse, une application centralisée au niveau du serveur n'a accès qu'à l'activité de ce dernier, indépendamment des environnements des utilisateurs concernés.

Notre travail se situe dans le cadre le plus général possible : en ce sens, nous voulons que nos méthodes s'appliquent quelque soit le site web utilisé et quelque soit la manière dont l'utilisateur y accède. Ainsi, nous verrons que certains types de données ont par

nature un coût d'acquisition trop important pour être utilisés dans un cadre général. L'acquisition de données exotiques pour une application spécifique peut être envisageable mais ne relève pas de ce travail : nous montrons qu'il est possible d'extraire des informations de navigation d'utilisateur pour améliorer les services web, indépendamment du site observé et des outils utilisés.

Le protocole HTTP est défini de manière bi-latérale (client-serveur) : nous étudions donc les différents types de données disponibles et leurs acquisitions des deux côtés, client puis serveur.

2.2 Côté client ⋆

L'acquisition des données du côté du client correspond au cas où l'utilisateur veut une meilleure qualité de services en autorisant le système à connaître toute son activité de navigation. Imaginons qu'à chaque utilisateur d'un site web soit associée une personne[1] qui noterait l'intégralité du comportement, du plus simple clic de la souris, à un clignement d'œil. Dans ce contexte, la diversité des données récupérables est immense. Nous décrivons ci-après quelques types de données utilisés dans des travaux similaires.

Dans [PPPS03], les auteurs expliquent que la connaissance de données côté client implique soit la mise en place d'une partie du système sur le navigateur du client, soit l'utilisation d'un navigateur spécialement dédié pour l'envoi d'information supplémentaire. La première technique, largement utilisée [BLP+03, SBKF01], fait souvent appel à une application en Java ou JavaScript exécutée par le client. Ce type d'interfaçage permet de récupérer tout ou partie des informations disponibles sur le navigateur : le type de système d'exploitation, le type de navigateur, les paramètres de ce dernier (langues préférées, résolution d'affichage, types d'application cliente reconnus), mouvements et clics de systèmes de pointage, entrées au clavier, *etc.* L'utilisation [CP95] d'un navigateur « enrichi » est peu utilisée à cause du développement demandé pour un tel système.

Récemment, un nouveau concept d'affichage des données web a été mis en place : il s'agit de faire plusieurs requêtes pour le contenu même de la page, en échangeant des données sous format Xml et/ou Html. En effet, le navigateur dit « enrichi » possède en plus d'un moteur d'affichage, un moteur de demande de requêtes (gérant le Xml). Ainsi le navigateur peut demander que les parties dont il a besoin pour afficher convenablement une page web, et même (et c'est là son grand intérêt), redemander une partie qui pourrait changer en fonction de l'interaction avec l'utilisateur. Cet ensemble de techniques associées, est dénommé Ajax[2] [Gar05]. Cette façon d'afficher les données demande une grande interaction entre moteur d'affichage et moteur de requête. Pour le moteur Gecko (cf. table 2.1), un langage dédié Xul a été défini pour interagir de manière simple avec le moteur d'affichage. Malheureusement, nous verrons que les navigateurs, même les plus récents, ne gèrent pas les normes DOM et ECMAScript

1 ou un système de surveillance avec des facultés sensorielles infaillibles

2 Acronyme pour Asynchronous Javascript + Xml

(norme de langage à l'origine de Javascript) de manière indentique. Un développement coûteux est donc nécessaire aves ces nouveaux clients « enrichis ».

Il est à noter que d'autres données ont pu être utilisées de manière marginale dans le but d'augmenter les pertinences de traitement. Ces dernières sont généralement issues de protocoles d'acquisition lourds et coûteux. Les données oculométriques [GE03], en sont un exemple : le mouvement des yeux des utilisateurs est enregistré par une caméra, ce qui permet de reconstituer le chemin visuel de l'utilisateur. Ces données sont plus souvent utilisées pour améliorer l'ergonomie d'un site web. En effet, la difficulté [GSL+02] inhérente à l'acquisition de ces dernières ne permet pas une exploitation simple pour un problème qui se veut général.

L'exécution côté client de code développé pour un serveur en particulier repose sur de nombreux concepts : en effet, pour ce type de mise en œuvre, il est indispensable de pouvoir appliquer une commande à une partie d'un document ou à un élément du navigateur (comme par exemple : changer le type de message de la barre d'état du navigateur). Pour cela, il est indispensable d'utiliser un langage de programmation agissant sur des objets définis comme étant soit des parties du document, soit du navigateur lui-même. Les différents navigateurs utilisent tous un langage de programmation compatible avec la norme ECMAScript [Int99] agissant sur des objets du DOM[3] [W3C04]. Un des problèmes, cependant, est la grande diversité des implantations des langages de type ECMAScript et du modèle d'objets propres à chaque client. Pour résumer, la table 2.1 rassemble les différentes entités utilisées pour chacun des six moteurs d'affi-

Moteur	Version ECMAScript	Version DOM[4]
Gecko, SpiderMonkey	JavaScript 1.6	DOM Level 3 (non testé)
Trident	JScript 5.6	DOM Level 3 (en partie) [5]
Presto	ECMAScript[6]	DOM Level 3
Tasman	JScript 5.6	DOM Level 0
iCab	InScript 3.22	DOM Level 2
KHTML	JavaScript 1.5	DOM Level 2

TAB. 2.1 – *Résumé des différents langages* ECMAScript *et modèles* DOM *supportés par les navigateurs courants*

chage les plus répandus. Notons que les navigateurs issus de Mozilla (Firefox, Camino) utilisent le moteur Gecko, Internet Explorer pour Windows le moteur Trident, Opera le moteur Presto, Internet Explorer pour Mac le moteur Tasman, iCab le moteur éponyme, et enfin Konqueror le moteur KHTML.

Cette table rassemble les diverses versions ECMAScript et DOM de ces moteurs. Il apparait difficile alors de développer des méthodes génériques lorsqu'un navigateur choisi utilise des langages spécifiques. Le grand nombre de navigateur web supportant

3 De l'anglais : Document Object Model

4 Le site http://www.webdevout.net/browser_support_dom.php donne en détail l'état de support des spécifications DOM. Les navigateurs de type Gecko ont l'air les plus avancés.

5 « With only a few minor exceptions » : annotation mentionnée sur le site de Microsoft

6 « more or less aligned with JavaScript 1.3/1.5 Core » : annotation mentionnée sur le site de Opera

ou pas telle ou telle spécification du W3C, ou l'implantant de manière non équivalente à un autre, nous conduit à ne pas envisager le côté client comme une bonne solution pour notre problème, à savoir récupérer de manière automatique les informations pertinentes de navigation d'un utilisateur.

En résumé, le côté client demande une acquisition des données fortement dépendantes de l'environnement où évolue l'utilisateur. Cette vision peut toutefois être envisagée pour une forte optimisation d'un site web particulier avec un coût élevé de développement. Placé du côté serveur, un système de surveillance sera moins coûteux mais ne verra que l'activité du serveur lui-même.

2.3 Côté serveur ⋆

À l'inverse du côté client où toutes les données ayant un lien avec la navigation de l'utilisateur pourraient être enregistrées (mais au prix d'un investissement colossal), le côté serveur ne dispose que de très peu de données, toutes regroupées dans les fichiers de logs et la structure même du site considéré.

2.3.1 *Logs* serveurs ⋆

L'activité d'un serveur web est composée d'une succession d'étapes : la réception d'une requête en provenance d'un client, l'analyse de la requête, la création de la réponse, l'envoi de cette dernière. La totalité de ces informations peut être stockée dans un fichier d'enregistrements (ou *logs*), mais en pratique seule une sélection d'informations propre à chaque serveur est sauvegardée. Le W3C recommande [Luo95] toutefois de stocker les éléments suivants :

- nom (ou adresse réseau) de la machine cliente ;
- nom d'utilisateur distant ;
- nom d'utilisateur authentifié ;
- date et heure de la requête ;
- requête envoyée par le client ;
- type de réponse du serveur ;
- taille de la réponse.

Le protocole `HTTP` prévoit lors de la création d'une requête (respectivement d'une réponse) une liste de données permettant de définir divers autres paramètres : de la gestion des caches à l'encodage de la requête (ou réponse) en passant par des informations complémentaires de nature variée (langage, type de navigateur, information supplémentaire d'erreur, date de dernière modification du document, type de contenu de la réponse). Ainsi une requête/réponse est toujours composée d'une première ligne mentionnant son type, suivie d'un nombre quelconque de lignes facultatives décrivant les paramètres optionnels.

Pour rechercher dans cette source de données des informations pertinentes sur la navigation d'un utilisateur, nous devons définir quelques notions de bases. Pour cela, le

W3C définit dans [W3C99c] les termes utilisés dans ce domaine. Ce dernier document est divisé en trois parties définissant les contextes dans lesquels les définitions sont valides : la différence est faite entre côté client, côté serveur et les ressources en elles-mêmes. La définition la plus importante concerne ce que nous appellerons une « visite » d'un utilisateur sur un site web donné. À juste titre le W3C définit ceci dans le contexte serveur, et l'appelle « session serveur » ou « visite ». Le terme « session » est ambigu et ne doit pas être rapproché de la session définie dans le contexte client : cette dernière est un sous-ensemble de toutes les pages vues (sur tous les serveurs) par le client.

Définition 2.1 (Visite) *Considérons un utilisateur U et un site web S, nous appelons* visite *de U sur S l'ensemble des pages vues par U sur le site S.*

> *Remarque :* La définition précédente reprend celle du w3c. Nous ajoutons à celle-ci une notion sémantique : nous appelons visite l'ensemble des pages demandées dans un but précis. Par exemple, dans le cas où un utilisateur recherche deux informations sur un même site, nous considérerons les deux ensembles de pages comme des visites distinctes.

Dans le but de retrouver des similitudes dans la navigation de l'usager d'un site web considéré, il nous parait essentiel de pouvoir définir comme unité de travail les visites utilisateur au sens défini précédemment. C'est donc en considérant le côté serveur du protocole `HTTP` que l'on peut avoir accès à ces fichiers de logs. Le chapitre suivant présente en détails les transformations nécessaires pour passer de fichiers de logs bruts à des visites utilisateurs facilement traitables.

L'algorithme décrit dans le chapitre suivant a besoin, pour traiter les incohérences des logs, de la structure même du site web, pour cela notre point de vue et de montrer le site comme un graphe.

2.3.2 Extraction de graphe sous-jacent ⋆

De manière générale, un site web peut être vu comme un graphe composé de sommets (pages web) reliées entre elles. Nous définissons ci-après ce que nous appelons un graphe, puis nous montrons le passage trivial du site web au graphe.

Définition 2.2 (Graphe) *Soit N un ensemble fini d'éléments $\{n_1,n_2,\ldots,n_x\}$, soit $A = \{(n_1,n_2),(n_2,n_1),(n_x,n_x)\}$ un ensemble de couples d'éléments de N, $G = (N,A)$ est un* graphe *orienté. Les éléments de N sont appelés* noeuds *ou* sommets*, les éléments de A* arcs*.*

Les graphes *ont la possibilité d'être représentés visuellement. Par exemple, la figure 2.1 montre un graphe composé de $|N| = 3$ nœuds, avec $|A| = 5$ arcs $\{(n_1,n_2)$, (n_2,n_3), (n_3,n_3), (n_3,n_1), $(n_1,n_3)\}$.*

> *Remarque :* On peut également définir les graphes non orientés : l'ensemble A est constitué non pas de couples d'éléments de N, mais d'ensembles de taille deux. Ainsi ces éléments sont appelés arêtes et il n'y a pas de différence entre l'arête $\{n_1,n_2\}$ et l'arête $\{n_2,n_1\}$, contrairement aux arcs.

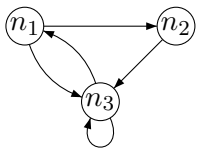


Fig. 2.1 – *Exemple de graphe*

Les pages interconnectées d'un site web peuvent être vues comme les nœuds d'un graphe. En interprétant les hyperliens reliant les pages entre elles comme des arcs, nous obtenons une structure de graphe. C'est ce que nous appelons le graphe *sous-jacent* d'un site web.

Donc en parcourant simplement les pages d'un site web, nous pouvons extraire la structure de graphe sous-jacente. L'algorithme présenté ci-après est un parcours infixe du site web. L'implantation de ce dernier ne présente pas de difficulté majeure, excepté d'être robuste aux pages ne respectant pas exactement les standards définis par le W3C. Le parcours est défini par l'algorithme 2.1.

L'algorithme 2.1 est une extension aux graphes d'un parcours simple en largeur d'abord d'un arbre. À chaque page du site vue, on associe un nœud, les hyperliens deviennent des arcs. Initialement le graphe est constitué uniquement de la page de démarrage (paramètre de l'algorithme) puis, de proche en proche, il s'étoffe, jusqu'à représenter le site entièrement accessible depuis cette page. La difficulté de cette extraction tient dans la fonction `Extraire_Liens` et la façon de gérer les pages contenant des cadres (*framesets*).

2.4 Conclusion ⋆

Ainsi l'architecture du protocole `HTTP` réserve du côté client, comme du côté serveur, des catégories de données disponibles renfermant des informations sur le comportement de l'utilisateur sur un site web. Du point de vue client, les données sont d'une extrême fiabilité mais sont dépendantes de l'environnement client et ont un coût important : le développement du client modifié, de l'application embarquée ou encore le protocole de génération des données représentent des investissements en temps très importants. À l'opposé, côté serveur, les données peuvent être moins fiables (cf. les diverses fonctionnalités réseau dans le chapitre 1 qui induisent une différence entre la navigation de l'utilisateur et celle enregistrée par le serveur), demandent un traitement important mais sont indépendantes de l'environnement client et du site web considéré. Nous nous orientions donc vers les données disponibles côté serveur, indépendemment des types de clients utilisés.

En faisant l'hypothèse de serveur web enregistrant — au minimum — les traces recommandés [Luo95] par le W3C, les logs sont la seule source de données indépendantes du serveur web et du type de client utilisé. En nous focalisant sur les informations

Algorithme 2.1 : Extraction_graphe

```
Données : S un site web, page_de_démarrage une page de démarrage
Résultat : G un graphe représentant le site web
G = Crée_Graphe({page_de_démarrage});
F = Crée_File({page_de_démarrage});
États_traités = Crée_Table_Hachage(page_de_démarrage => 1);
tant que F ≠ ∅ faire
    page_courante = Défile(S);
    Ouvrir(page_courante);
    ligne_courante = Lire_Ligne(page_courante);
    tant que ! Fin(ligne_courante) faire
        si Liens = Extraire_Liens(ligne_courante) alors
            pour chaque lien ∈ Liens faire
                Ajoute_Fils(lien, page_courante, G);
                si États_traités(lien) n'existe pas alors
                    États_traités(lien) = 1;
                    Enfile(lien, S);
                fin
            fin
        fin
        ligne_courante = Lire_Ligne(page_courante);
    fin
    Fermer(page_courante);
fin
retourner G;
```

contenues dans les logs et sur l'architecture du graphe représentant le site web considéré, nous finissons l'étape de sélection dans le processus d'extraction de connaissance, et nous présentons dans la suite le traitement de ces données.

3 Logs ⋆

Sommaire

Résumé

Dans ce chapitre, nous définissons ce que sont les logs enregistrés sur un serveur web. Nous introduisons les difficultés de leur utilisation sans prétraitement. La structure des documents hypertextes et certaines particularités des protocoles réseaux empêchent une correspondance exacte entre enregistrement et navigation. Pour chacune de ces raisons, nous présentons en détail le type de problème induit, son influence sur le processus complet d'extraction de connaissance, et les solutions que nous adoptons pour gérer ce dernier.

3.1 Introduction ⋆

Les serveurs web, en tant que logiciels servant des pages web, enregistrent, souvent à des fins de sécurité ou de métrologie statistique, les différentes événements représentant leur activité de communication. Grâce à cela, une trace simple est conservée des communications passées. L'analyse de ces traces doit nous permettre d'extraire les informations pertinentes concernant les utilisateurs du site en question. En première approche, nous pouvons mentionner qu'il existe nombre d'utilitaires commerciaux [Bou] ou libres [Bar] analysant de manière superficielle ces logs pour en extraire des indicateurs simples : le nombre de requêtes hebdomadaires ou quotidiennes, la quantité de données physiques reçue et envoyée sur le réseau, les types de réponses du serveur les plus utilisées, les pages les plus demandées, *etc*.

Les *logs* (ou enregistrements) se présentent sous la forme de fichiers où chaque ligne est composée de champs sur la requête et la réponse à celle-ci. Le tableau 3.1 montre un exemple de logs enregistrés sur un serveur web. Cet exemple reprend uniquement les champs recommandés du W3C ; dans le reste du document, nous faisons l'hypothèse que nous ne disposons que de ces champs dans les logs. Le premier champ, M_1, représente

M_1 `- - [30/Oct/2001:20:13:27 +0100] "GET /`A` HTTP/1.1" 200 293`

TAB. 3.1 – *Exemple de fichier de logs*

le nom de la machine ayant émis la requête ; dans le cas où le nom est indisponible, l'adresse réseau est renseignée : cette dernière est toujours disponible du fait de la nature du protocole `HTTP`. Le champ suivant, `-`, correspond au nom d'utilisateur distant, c'est-à-dire le nom de connexion de l'utilisateur sur la machine distante ; le tiret indique ici que la valeur est très souvent non renseignée, en effet pour des questions de sécurité, les noms de connexion ne doivent pas être divulgués partout sur le réseau, de plus les machines intermédiaires (*proxy*, *reverse-proxy*, etc.) ne gèrent pas forcément bien ce paramètre. De même, le tiret suivant est uniquement renseigné lors de l'utilisation du protocole d'authentification intégré dans le protocole `HTTP` ; dans le cas général l'authentification n'est pas obligatoire et donc le champ non renseigné. La date de la requête suit ensuite dans la liste des champs : elle est composée de la date (jour, mois, année) et de l'heure de la requête à la seconde près, ainsi sur un serveur plusieurs requêtes peuvent avoir lieu avec la même date enregistrée, c'est l'ordre chronologique d'apparition dans le fichier d'enregistrements qui permet de les classer plus finement. L'information qui suit est la requête, ou plus exactement la première ligne de la requête émise par le client : nous avons vu précédemment (section. 2.3.1) qu'une requête est composée d'une ligne de requête suivie d'un nombre quelconque (voire nul) d'éléments additionnels, seule la ligne de requête est sauvegardée et par soucis de simplification nous la noterons souvent requête l'élément signifiant ligne de requête. Le champ suivant, ici `200`, définit le code de retour [1] ; il correspond à la façon dont le serveur gère la requête du client. Enfin, `293` est le dernier champ et représente la taille en octets de la réponse adressée par le serveur à la machine cliente.

L'amélioration automatique des services du *Web* pour un utilisateur donné (ou un groupe d'utilisateurs ayant sensiblement le même type de navigation) repose sur la possibilité de disposer d'informations répertoriées par utilisateur et non chronologiquement en fonction des requêtes reçues. De plus, ces données doivent être les plus fiables possibles en représentant au mieux la navigation de la personne concernée. En cela les outils génériques et basiques d'analyses présentés précédemment ont des limites : en effet, la méthode générale employée ne prend pas en compte les problèmes de données incohérentes ou manquantes ni l'architecture du site web considéré.

Pour éliminer ces problèmes inhérents aux données sélectionnées, nous devons prétraiter ces dernières pour les rendre utilisables dans le processus d'inférence grammaticale que nous avons choisi d'utiliser. Cette étape regroupe les phases de nettoyage et de transformation du processus global d'extraction de connaissance. Le choix de l'inférence grammaticale pour apprendre automatiquement un modèle d'utilisateur avec ce type de données est cohérent seulement si la fiabilité des données est réelle. C'est pour cela que nous nettoyons les logs en supprimant les données incohérentes et inutiles d'une part, et en restructurant les données de manière à ce qu'elles reflètent parfaitement la

1 Le lecteur pourra se rendre à la section 1.4 pour les définitions.

navigation de l'utilisateur d'autre part. Enfin, nous décrivons l'étape « transformation » qui réorganise les *logs* en visites.

3.2 Nettoyage ⋆

Le nettoyage des données vise à éliminer toutes les requêtes considérées comme inutiles de l'ensemble de logs de départ. En effet, une quantité non négligeable des enregistrements d'un serveur web ne correspond à aucune ressource valide. De plus, les fonctionnalités du réseau décrites dans la section 1.6 dégradent les enregistrements du serveurs, car certaines requêtes n'aboutissent pas et d'autres sont faussées. La phase de nettoyage s'effectue en ayant connaissance du graphe correspondant au site web considéré grâce à une lecture séquentielle des enregistrements ; nous expliquons donc les problèmes en détail et ce que nous apportons pour leur gestion (cf. algorithme 3.1).

3.2.1 Données inutiles ou incohérentes ⋆

Le protocole `Http`, précédemment défini, permet de disposer des ressources de tout type : page web, élément multimédia, programme, donnée quelconque. Lors d'une requête correspondant à une page intégrant d'autres ressources (généralement des images, ou de petites animations), le client exécute effectivement plusieurs requêtes vers le serveur : une pour la page (le contenant), une pour les divers éléments (les contenus). Ainsi, pour une page demandée, plusieurs requêtes peuvent aboutir au serveur. En se référant au but de notre travail, à savoir l'extraction de modèle d'utilisateurs pour l'amélioration de services web, il nous parait judicieux de ne conserver que les pages web (dites contenant), sans les éléments incorporés. Évidemment, sur des applications visant à extraire le comportement d'utilisateurs de site web de bibliothèques d'images en ligne, ce type de ressources doit être conservé, mais dans le dessein d'un procédé général, nous simplifions les données des logs en enlevant tout ce qui ne correspond pas à une page web. Dans l'idéal, ce filtrage des requêtes devrait ce faire sur le type de contenu répondu (un champ supplémentaire dans la définition de `Http`, cf. Sect. 2.3.1), mais l'hypothèse que nous avons faite sur les champs présents dans les logs, nous oblige à filtrer le type de page demandé, plus exactement à effectuer un filtrage sur l'extension de la page demandée (nous supprimons les extensions connues d'images et autres composants multimédia).

De plus, nous faisons le choix de ne garder pour la phase d'apprentissage que les requêtes ayant abouti, c'est-à-dire correspondant à une ressource valide. Le code de retour présent dans les logs nous permet de filtrer ces requêtes. Par conséquent, nous ne gardons que celles de code égal à 200.

Des données dites incohérentes peuvent subsister : en effet, lors du traitement des logs, chaque page est mise en correspondance avec le nœud du graphe sous-jacent du site web considéré. Ce graphe n'est malheureusement qu'un instantané du site en question qui a pu être modifié. Ainsi, certaines requêtes contenues dans les logs peuvent ne pas correspondre à des pages du site actuel : ces requêtes ne sont pas prises en compte.

Un dernier type de requête, non seulement inutiles, mais apportant un bruit dans les données, correspond à celles effectuées de manière automatique par un « robot ». Il existe plusieurs fonctionnalités dites d'optimisation de la vitesse de navigation qui induisent des requêtes inutiles pour la navigation et donc ne reflétant pas le comportement de l'utilisateur. Ces processus sont de deux sortes : les robots (ou « araignées » en référence à la toile, signification de *Web* en anglais) et les systèmes de *prefetching*. Les moteurs de recherche utilisent tous des robots plus ou moins sophistiqués qui parcourent la totalité[2] du site web considéré. Ces requêtes n'indiquent en rien un comportement utilisateur, nous nous efforçons de ne pas les prendre en compte : une page web [VG] recense les machines de chacun des robots des moteurs de recherche. Ces données libres nous permettent là encore de filtrer les requêtes, et de supprimer celles qui sont indésirables. Les systèmes de *prefetching* tentent de demander au serveur une page avant que l'utilisateur en fasse une demande explicite : ainsi la page demandée par le système automatique se situe dans le cache et est disponible instantanément. Ces systèmes sont souvent embarqués dans les navigateurs clients (Mozilla, Firefox).

Nous venons de voir que les logs contiennent des données superflues qui sont inutiles ou incohérentes, voire incorporant du bruit. Une phase de filtrage permet de les éliminer. Nous considérons maintenant les requêtes qui relèvent de la navigation d'un utilisateur mais qui n'ont pas abouti au serveur.

3.2.2 Données manquantes ⋆

Les fichiers de logs, pour de multiples raisons, peuvent ne pas refléter exactement la navigation des utilisateurs sur un site web. Ainsi, une partie des demandes d'un client n'arrive jamais au serveur, une réponse est envoyée par une machine intermédiaire, ou directement via un système de mise en cache local.

Nous étudions ci-après les différentes causes pouvant aboutir à un manque de données, et nous les illustrons par un exemple.

Considérons un site web composé de huit pages web liées suivant la représentation en graphe définie par la figure 3.1. Les nœuds (ou pages) représentés de manière grisée ne peuvent être mis en cache (utilisation des directives `HTTP` prévues pour cela). Considérons deux utilisateurs accédant à ce site web depuis deux machines distinctes $M_{\{1,2\}}$. En notant P_i, la requête de la page P par la machine M_i, nous supposons que le premier (respectivement second) utilisateur effectue une visite sur le site en visualisant, dans cet ordre, les pages A-C-E-C-A-D-G-H (resp. B-D-F-D-F-D-A-C-E). Supposons maintenant qu'ils accèdent au serveur de manière simultanée, l'ordre des requêtes est le suivant : A_1-C_1-B_2-D_2-F_2-E_1-C_1-A_1-D_1-G_1-H_1-D_2-F_2-D_2-A_2-C_2-E_2. Sans fonctionnalités réseau additionnelles, en supposant que les clients sont directement connectés au serveur, et qu'il n'existe pas de système de cache local, les enregistrements dans les fichiers de logs ressembleraient à ceux du tableau 3.2.

[2] Certains robots autorisent un parcours spécifique et propre à chaque serveur, grâce à un fichier de configuration.

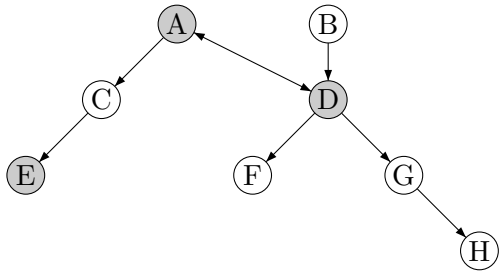


FIG. 3.1 – *Exemple de site web*

Nous étudirons par la suite les diverses causes de données manquantes en illustrant les problèmes occasionnés sur cette même série de requêtes.

Les causes de pertes de données

La mise en cache local Les navigateurs web actuels intègrent souvent un système de mise en cache local. Ce dernier est souvent composé de deux parties : une en mémoire centrale (pour un accès rapide), l'autre en mémoire disque (pour garder une grande quantité de données). Les divers systèmes de mise en cache sont plus ou moins développés en permettant notamment à l'utilisateur de modifier les paramètres classiques : temps de garde d'un document, type de document à cacher, etc. Les algorithmes de mise en cache classiques dépassent l'objet de ce travail et nous invitons le lecteur intéressé à lire [ASA+95, CDN+96].

Les pages demandées initialement par l'utilisateur sont ainsi disponibles dans le cache. Ensuite, une requête pour une page déjà cachée n'aboutira pas au serveur, le cache relayant l'information à sa place. Ainsi, certaines requêtes ne peuvent être enregistrées sur le serveur.

En reprenant l'exemple développé précédemment, et en simulant l'activité d'un cache local sur chaque machine, l'enregistrement des logs correspondrait à celui représenté dans le tableau 3.3

Il est évident que, placé du côté serveur, un système automatique d'extraction de modèles utilisateurs ne pourra pas avoir accès à certaines requêtes effectuées mais non abouties au serveur : dans l'exemple deux requêtes sont ainsi touchées. La deuxième requête de la page C depuis la machine M_1 et la deuxième requête de la page F depuis la machine M_2 ne pourront être enregistrées. Nous rappelons que les pages A et D, qui devraient être impactées, ne le sont pas car elles sont définies comme non cachables.

Ceci représente en effet la première source de données manquantes ou erronées. Mais, sur la route entre un client et le serveur, il peut y avoir plusieurs machines relais qui perturbent encore les fichiers de logs.

```
M1 - - [30/Oct/2001:20:13:27 +0100] "GET /A HTTP/1.1" 200 293
M1 - - [30/Oct/2001:20:14:27 +0100] "GET /C HTTP/1.1" 200 273
M2 - - [30/Oct/2001:20:14:55 +0100] "GET /B HTTP/1.1" 200 148
M2 - - [30/Oct/2001:20:14:57 +0100] "GET /D HTTP/1.1" 200 159
M2 - - [30/Oct/2001:20:14:59 +0100] "GET /F HTTP/1.1" 200 171
M1 - - [30/Oct/2001:20:15:00 +0100] "GET /E HTTP/1.1" 200 612
M1 - - [30/Oct/2001:20:15:03 +0100] "GET /C HTTP/1.1" 200 273
M1 - - [30/Oct/2001:20:15:31 +0100] "GET /A HTTP/1.1" 200 293
M1 - - [30/Oct/2001:20:15:32 +0100] "GET /D HTTP/1.1" 200 159
M1 - - [30/Oct/2001:20:15:37 +0100] "GET /G HTTP/1.1" 200 631
M1 - - [30/Oct/2001:20:15:41 +0100] "GET /H HTTP/1.1" 200 423
M2 - - [30/Oct/2001:20:15:59 +0100] "GET /D HTTP/1.1" 200 159
M2 - - [30/Oct/2001:20:16:15 +0100] "GET /F HTTP/1.1" 200 171
M2 - - [30/Oct/2001:20:16:17 +0100] "GET /D HTTP/1.1" 200 159
M2 - - [30/Oct/2001:20:16:40 +0100] "GET /A HTTP/1.1" 200 293
M2 - - [30/Oct/2001:20:17:00 +0100] "GET /C HTTP/1.1" 200 273
M2 - - [30/Oct/2001:20:17:05 +0100] "GET /E HTTP/1.1" 200 612
```

TAB. 3.2 – *Fichier de logs : le cas idéal*

Un *proxy* intermédiaire La mise en place d'un *proxy* est fréquente dans les structure ou nombre de personnes accèdent à l'internet. Pour des raisons d'administration réseau sécurisée, il parait logique de gérer les flots web entre l'entité (souvent plusieurs centaines d'utilisateurs) et le web, en forçant le passage par une machine intermédiaire (située dans la structure, côté client), seule à disposer d'un accès au *Web*.

De plus, pour des raisons d'efficacité, la machine placée entre les acteurs de la communication est presque toujours couplée à un système de mise en cache. Ce dernier ne concerne plus uniquement un seul utilisateur (cas du cache dit « local ») mais tous les utilisateurs de l'entité : on parle alors de cache « global ».

Les pages demandées par n'importe quel utilisateur de l'entité peuvent ensuite être directement servies par le cache pour toute demande de la même page, éventuellement d'un autre utilisateur.

En reprenant l'exemple développé précédemment, et en simulant l'activité d'un cache global sur une machine de type *proxy*, l'enregistrement des logs correspondrait à celui représenté dans le tableau 3.4

En se plaçant toujours du côté serveur, de la même façon qu'avec les seuls caches locaux, le système ne pourra pas avoir accès à certaines requêtes effectuées mais non abouties au serveur : en reprenant notre exemple, deux requêtes supplémentaires sont ainsi touchées. La première requête de la page C (respectivement E) par M_2 sera servie par le *proxy* — M_1 l'ayant auparavant demandée. De plus, nous constatons aussi que l'adresse de provenance des requêtes qui arrivent ne correspond plus à une machine d'un utilisateur mais à la machine intermédiaire : nous obtenons ainsi des données erronées par rapport aux modèles utilisateurs avec deux utilisateurs ayant la même adresse de provenance. Un système automatique basique ne verrait qu'un seul utilisateur sur le

```
M1 - - [30/Oct/2001:20:13:27 +0100] "GET /A HTTP/1.1" 200 293
M1 - - [30/Oct/2001:20:14:27 +0100] "GET /C HTTP/1.1" 200 273
M2 - - [30/Oct/2001:20:14:55 +0100] "GET /B HTTP/1.1" 200 148
M2 - - [30/Oct/2001:20:14:57 +0100] "GET /D HTTP/1.1" 200 159
M2 - - [30/Oct/2001:20:14:59 +0100] "GET /F HTTP/1.1" 200 171
M1 - - [30/Oct/2001:20:15:00 +0100] "GET /E HTTP/1.1" 200 612
M1 - - [30/Oct/2001:20:15:03 +0100] "GET /G HTTP/1.1" 200 273
M1 - - [30/Oct/2001:20:15:31 +0100] "GET /A HTTP/1.1" 200 293
M1 - - [30/Oct/2001:20:15:32 +0100] "GET /D HTTP/1.1" 200 159
M1 - - [30/Oct/2001:20:15:37 +0100] "GET /G HTTP/1.1" 200 631
M1 - - [30/Oct/2001:20:15:41 +0100] "GET /H HTTP/1.1" 200 423
M2 - - [30/Oct/2001:20:15:59 +0100] "GET /D HTTP/1.1" 200 159
M2 - - [30/Oct/2001:20:16:15 +0100] "GET /F HTTP/1.1" 200 171
M2 - - [30/Oct/2001:20:16:17 +0100] "GET /D HTTP/1.1" 200 159
M2 - - [30/Oct/2001:20:16:40 +0100] "GET /A HTTP/1.1" 200 293
M2 - - [30/Oct/2001:20:17:00 +0100] "GET /C HTTP/1.1" 200 273
M2 - - [30/Oct/2001:20:17:05 +0100] "GET /E HTTP/1.1" 200 612
```

TAB. 3.3 – *Fichier de logs : l'influence du cache local*

site web, dans le cas présent.

Un *reverse-proxy* intermédiaire Un autre type de machine placée entre le client et le serveur, qui sert à effectuer une mise en cache inversée, est le *reverse-proxy*. En effet, comme le *proxy* permettait un accès « sortant » sur le *Web* uniquement *via* une seule machine pour des raisons de sécurité et de performances, le *reverse-proxy* sert quant à lui à restreindre l'accès « entrant » à des serveurs, pour les mêmes raisons (côté serveur). Le type de problème engendré est flagrant : les logs ne contiennent plus que des requêtes provenant de cette machine. Dans le problème qui nous intéresse, nous nous plaçons côté serveur et donc nous ne prenons pas en compte les effets d'un *reverse-proxy*, car nous avons la possibilité de l'enlever de la chaîne, contrairement aux autres machines intermédiaires. Cet élément est donc présenté ici par souci de complétude car il perturbe les enregistrements de logs sur le serveur. Nous invitons le lecteur intéressé à se documenter sur le plus grand réseau de *reverse-proxy* : Akamai[Aka].

Nous avons vu pourquoi les données enregistrées dans les logs du serveur peuvent ne pas représenter la réalité des utilisateurs : premièrement, il peut manquer des requêtes — ceci est dû à la mise en cache local ou global ; deuxièmement, l'adresse de provenance peut être erronée et regrouper en réalité plusieurs utilisateurs.

Solutions

Les données manquantes ou erronées correspondent à une utilisation générale du *Web*, en effet l'architecture client, cache, proxy-cache, serveur est très répandue. Il existe

```
proxy - - [30/Oct/2001:20:13:27 +0100] "GET /A HTTP/1.1" 200 293
proxy - - [30/Oct/2001:20:14:27 +0100] "GET /C HTTP/1.1" 200 273
proxy - - [30/Oct/2001:20:14:55 +0100] "GET /B HTTP/1.1" 200 148
proxy - - [30/Oct/2001:20:14:57 +0100] "GET /D HTTP/1.1" 200 159
proxy - - [30/Oct/2001:20:14:59 +0100] "GET /F HTTP/1.1" 200 171
proxy - - [30/Oct/2001:20:15:00 +0100] "GET /E HTTP/1.1" 200 612
M1 - - [30/Oct/2001:20:15:03 +0100] "GET /C HTTP/1.1" 200 273
proxy - - [30/Oct/2001:20:15:31 +0100] "GET /A HTTP/1.1" 200 293
proxy - - [30/Oct/2001:20:15:32 +0100] "GET /D HTTP/1.1" 200 159
proxy - - [30/Oct/2001:20:15:37 +0100] "GET /G HTTP/1.1" 200 631
proxy - - [30/Oct/2001:20:15:41 +0100] "GET /H HTTP/1.1" 200 423
proxy - - [30/Oct/2001:20:15:59 +0100] "GET /D HTTP/1.1" 200 159
M2 - - [30/Oct/2001:20:16:15 +0100] "GET /F HTTP/1.1" 200 171
proxy - - [30/Oct/2001:20:16:17 +0100] "GET /D HTTP/1.1" 200 159
proxy - - [30/Oct/2001:20:16:40 +0100] "GET /A HTTP/1.1" 200 293
M2 - - [30/Oct/2001:20:17:00 +0100] "GET /C HTTP/1.1" 200 273
M2 - - [30/Oct/2001:20:17:05 +0100] "GET /E HTTP/1.1" 200 612
```

TAB. 3.4 – *Fichier de logs : l'influence du cache global*

plusieurs autres domaines d'utilisation où ces types de problèmes ne se posent pas, ou peu. Nous étudions ci-après cinq domaines d'utilisation différents du *Web*, et enfin nous nous intéressons à une méthode de détection d'erreurs et de données manquantes en régime général.

Si le domaine d'application lié à l'architecture client, cache, proxy-cache ou serveur pose problème, il est naturel d'essayer de se placer dans des situations différentes et de voir si cela n'impose pas de trop fortes contraintes quant à l'application de ces méthodes sur un serveur considéré comme étant le plus général possible.

Le travail en local La première idée, et la plus simple, est de construire une quantité d'enregistrements depuis des clients directement connectés au serveur (sans proxy, voire sans cache). Cette stratégie, bien qu'envisageable pour une optimisation pointue d'un site web d'une grande et riche structure, est en réalité non acceptable pour notre problème. En effet, l'acquisition de quantité suffisante — dépendante de la taille et de la structure du site — de logs semble un problème très coûteux : il faut réaliser un ensemble de séances de navigations avec des utilisateurs/testeurs qui se connectent directement depuis l'entité hébergeant le serveur considéré sans passer par un proxy.

Le *cache-busting* Cette technique consiste à désactiver tout ou partie de la mise en cache. Les différentes versions du protocole HTTP gèrent de manière distincte la mise en cache : la version 1.0 définissant la directive PRAGMA autorise uniquement l'activation ou la désactivation totale du cache ; la version 1.1 permet une gestion plus fine en définissant une directive plus élaborée CACHE-CONTROL. Cette dernière méthode

permet de gérer le cache sous plusieurs aspects (cf. [W3C99b]) :

- des restrictions concernant ce qu'il est possible de mettre en cache ;
- des restrictions sur ce qu'il est possible d'enregistrer ;
- des modifications du système d'expiration du cache ;
- des mécanismes de contrôle sur la validation et le rechargement des pages ;
- des mécanismes de contrôle sur transformation des entités (certains *proxy-cache* peuvent par exemple stocker les images sous un format « rentable » en terme d'espace) ;
- des extensions du système.

La méthode consiste donc à arrêter les mises en cache sur le chemin de communication entre le client et le serveur. En activant les directives du protocole lors de l'envoi des données par le serveur, les machines intermédiaires et les caches locaux deviennent inactifs. En cela, le problème des requêtes manquantes n'existe plus, seul subsiste le problème de l'adresse de provenance erronée.

Deux raisons expliquent pourquoi cette méthode n'est pas envisageable sur le long terme : la déontologie et la fiabilité ne seraient pas respectées. En effet, la technique revient à passer outre les mises en cache, et donc à augmenter artificiellement le trafic réseau, ce qui n'est pas acceptable. De plus, les machines intermédiaires n'implantent pas toutes — comme elles le devraient — les directives de cache définies par HTTP. Ainsi une partie des données sera récupérée en augmentant les échanges web, et une autre ne le sera pas car les machines concernées n'obéiront pas aux directives du protocole ; sans aucune distinction entre ce qui est fiable ou pas.

Nous ne pouvons donc pas utiliser cette méthode telle qu'elle vient d'être présentée.

Les *cookies* Le terme anglophone *cookie* fait référence à une chaîne de caractères générée par le serveur en réponse à une requête initiale d'un client, qui est renvoyé au serveur à chaque nouvelle requête. En effet, le processus est défini ainsi : un « nouveau » client se connecte au serveur et demande une page web ; le serveur répond en envoyant un *cookie* sur le client suivi de la réponse à la requête. À chaque nouvelle requête, le client joint à sa demande le *cookie*. L'intérêt de cet outil et de pouvoir, avec une grande probabilité, classer les requêtes suivant les utilisateurs. Ce procédé présente des problèmes majeurs : la configuration des caches intermédiaires doit être compatible, les *cookies* sont définis par client de navigation et pour une durée définie à l'avance, et le fait de suivre virtuellement les déplacements d'un utilisateur sur un site peut poser des problèmes par rapport à la vie privée (pour ce point nous invitons le lecteur à se reporter à la section 1.7.1).

L'envoi systématique de *cookie* au serveur n'est possible uniquement que si les machines intermédiaires remplissent deux conditions : d'une part, elles doivent relayer les requêtes contenant un *cookie* en les présentant toujours au serveur — la mise en cache n'est donc pas possible pour ces requêtes — et, d'autre part, elles doivent présenter au serveur une requête complète — certains proxy-caches, pour des questions d'efficacité, simplifient les requêtes à la forme la plus simple possible, à savoir une simple URI demandée. L'arrêt de la mise en cache des requêtes est une source d'augmentation du

trafic réseau (cf. paragraphe précédent) : initialement, les requêtes devaient être à peine plus volumineuses de quelques octets (le *cookie* embarqué) mais au final c'est toute la communication entre le client et le serveur qui ne peut plus être mise en cache.

De plus, lors de la reconnaissance d'un type d'utilisateur, ce procédé peut amener certaines erreurs dans la détection : un *cookie* est défini sur un navigateur client pour une durée déterminée, soit en terme de temps fixe (20 minutes), soit jusqu'à la fermeture du navigateur. Dès lors, si l'utilisateur ferme son client ou change de machine, il sera vu comme un nouvel arrivant différent du précédent.

Pour toutes ces raisons, le principe de *cookie* systématique ne parait pas une bonne proposition pour la détection des utilisateurs d'un site web.

le *sampling* Le terme *sampling* (échantillonage) fait référence au domaine des statistiques. Dans [Pit97], PITKOW présente plusieurs méthodes qui consistent à effectuer du *cache-busting* partiel : en effet, en s'appuyant sur les théories statistiques, il définit certaines méthodes pour récupérer les données pouvant être manquantes dans les logs, avec un faible sur-coût de trafic réseau. Le principe est donc de ne pas mettre en cache les pages tout le temps, mais seulement sur des plages sélectionnées précisément. Ces sélections peuvent évidemment porter sur le temps — dans ce cas, le *cache-busting* n'est actif que sur certaines plages de temps —, ou encore sur les adresses de provenance ou avec une utilisation conjointe des cookies sur les personnes elles-mêmes. Ainsi, sur un échantillon de données « fiables et complètes », il est possible de déduire les propriétés sur la population entière des données avec une forte probabilité. En partant d'un grand ensemble de données imparfaites et d'un petit ensemble de données parfaites (sous l'hypothèse que le modèle de dégradation est le même pour toutes les données), l'obtention de l'ensemble total non perturbé est assuré. Le problème de ces techniques et de spécifier quelles doivent être les plages de temps, les adresses de provenance ou bien encore les personnes à sélectionner pour la partie de récupération sans mise en cache.

Suivant le type de paramètre, le choix peut être assez difficile, typiquement la distribution des adresses de provenance étant méconnue et vraissemblablement très loin d'une loi uniforme, il est impossible de définir à l'avance un sous ensemble qui sera représentatif de la population totale si nous voulons garder un échantillon assez petit.

De même, dans le cas où le paramètre retenu est le temps, la sélection des plages pour l'obtention d'un échantillon parfait n'est pas triviale. En effet, il est aisément concevable que la fréquentation d'un site web particulier ne soit pas du tout uniforme, et qu'un modèle représentant cette dernière soit difficile à trouver. Ainsi, quelles que soient les plages de temps sélectionnées, il est impossible d'assurer que l'échantillon des logs non cachés soit assez « proche » de la population réelle.

De plus, comme pour le *cache-busting* total, la fiabilité concernant le respect des machines intermédiaires à ne pas mettre en cache les données n'est pas assurée. Une telle technique n'est pas envisageable dans le cas général qui nous intéresse.

La gestion d'Uri dynamiques ou de sessions côté serveur Pour savoir à coup sûr qu'une page est demandée par la même personne (ou au moins le même navigateur) que la page précédente, des techniques existent et utilisent un même principe : ne pas envoyer plus d'une fois la même information en réponse à une même requête. En effet, chaque lien contenu initialement dans la page est codée de manière unique pour chaque requête, ce qui permet une désambiguïsation à terme dans les logs. Une telle pratique existe sur nombre de sites commerciaux.

Si une page est demandée plusieurs fois, par définition de ces méthodes, l'information renvoyée est différente : la mise en cache est donc impossible. Cela revient donc à faire du *cache-busting* classique. De plus, l'information disponible peut ne plus être identifiable par son adresse sur le *Web* ; en effet, l'Uri changeant à chaque demande, la ressource n'a plus une unique adresse physique. Enfin, la fiabilité est là aussi mise en doute : la façon de coder les Uri en fonction de la requête tient souvent dans la concaténation d'un numéro de session serveur à la fin de celle-ci (après le caractère ? ou &) ; dans ce cas, certains proxy-caches détectent que l'adresse n'a pas changé car le début est identique et que les paramètres suivant des caractères spéciaux (? et &) ne sont pas pris en compte.

> *Remarque :* La plupart des sites web dits « dynamiques », c.-à-d. qui exécutent du code côté serveur avant de servir la page demandée en fonction de la requête, utilisent des techniques de ce type. La mise en cache est donc impossible.

La détection d'incohérences dans les logs Enfin la détection des données manquantes peut être faite directement *via* les logs en repérant les incohérences dans le parcours de l'utilisateur. Cette technique est basée sur la connaissance *a priori* de la structure de graphe sous-jacente au site web considéré et à des heuristiques classant les logs dans la visite du bon utilisateur. Dans la section 3.3, ci-après, nous présentons en détail cette méthode qui cumule les avantages : le trafic réseau reste inchangé, le développement est fait une seule fois pour n'importe quel type de site web, la compatibilité est importante car la méthode se base sur les paramètres des logs recommandés par le W3C.

3.3 Reconstruction et transformation des logs ⋆

Comme nous l'avons déjà vu, les logs d'un serveur web contiennent des données sur la réponse que fait le serveur à une requête d'un client. De manière brute, seule l'adresse de provenance de la requête (adresse Ip du client) permet de classer les demandes en fonction de l'émetteur. Nous avons présenté aussi les problèmes liés au réseau induisant des données manquantes ou erronées. Nous présentons ci-après une méthode générale permettant de reconstruire certaines données manquantes et ainsi de transformer ces logs bruts en visite utilisateur constituée de la suite des pages vues. Ces séquences sont ensuite passées aux algorithmes d'inférence grammaticale pour en extraire un modèle de langage.

3.3.1 Visites utilisateur

Ces visites définies par le W3C sont très importantes : elles nous permettent de définir une recherche/navigation d'un utilisateur. Ce sont ces dernières (ou plutôt la séquence de page qu'elles représentent) qui serviront d'entrées aux algorithmes d'apprentissage.

À cause de certaines fonctionnalités réseau, nous avons vu qu'il était difficile de dire si une requête appartient à tel ou tel utilisateur, ou encore si l'utilisateur a vu des pages dont les requêtes n'ont pas abouti au serveur. Ainsi, le découpage en visites des logs pose quelques problèmes :

la durée Les logs ne contenant que des demandes de pages web, il n'existe pas de marqueur définissant la fin d'une visite.

le propriétaire Nous avons vu qu'un *proxy* pouvait masquer l'adresse de l'utilisateur au profit de sa propre adresse et ainsi rendre anonyme la requête : nous devons donc découvrir la visite à laquelle appartient le log.

les données manquantes En détectant les incohérences de navigation d'un même utilisateur, il est possible de rétablir certaines requêtes n'ayant pas abouti.

Ci-après nous présentons l'algorithme de reconstruction des logs basé sur la structure de graphe du site web considéré et la chronologie de l'enregistrement des requêtes dans les fichiers de logs.

3.3.2 Reconstruction des logs ⋆

L'algorithme 3.1 présente une méthode générale de grande compatibilité. Cette méthode ne prend en compte que les champs des logs dont l'enregistrement est recommandé par le W3C et la structure de graphe du site dont l'extraction est présenté dans l'algorithme 2.1.

Pour pouvoir présenter les heuristiques mises en place, nous en définissons les grandes lignes et indiquons quelques concepts mis en œuvre. Le principe repose sur la lecture séquentielle des logs enregistrés côté serveur, et place chaque réponse lue et valide dans la session la plus probable suivant divers critères de cohérence de navigation.

Le principe de l'algorithme est simple : il s'appuie sur le fait que la quasi-totalité des données manquantes sont des pages déjà vues par l'utilisateur. Lors d'une navigation, le principe de retour arrière sur une page déjà vue est fréquent, et la page étant cachée, la requête n'aboutit pas. Ainsi, pour une ligne de fichier de logs, il faut extraire la page demandée et l'affecter à la visite la plus probable. Pour faire ceci, nous cherchons dans les visites voisines dans le temps celles dans lesquelles il existe une page dite « source » permettant d'atteindre la page à classer. De manière plus formelle, si nous notons p la page à classer, nous cherchons une page q dans le graphe du site web $G = (N,A)$, telle que $(q,p) \in A$. Si cette page existe au moins dans une des visites, de manière intuitive, nous affectons la page à classer à la visite pour laquelle il a fallu remonter le moins loin dans l'historique pour trouver cette page source. De manière formelle, en reprenant les notations de l'algorithme, la visite v choisie est telle que $(\texttt{Visites}[v][h],p) \in A$ et $h = \min\{i \in \mathbb{N} | (\texttt{Visites}[v][i],p) \in A\}$. Dans le cas où aucune page source n'a été trouvée, une nouvelle session est créée : dans ce contexte, nous avons une page

Algorithme 3.1 : Reconstruction des Logs

```
Données : fichier_de_logs, un fichier de logs serveur
Résultat : Visites, un ensemble de visites utilisateur /* chaque élément est
   un tableau de pages, Visites[n][1] représente la dernière page
   incorporée dans la n^e visite ouverte                                  */
Visites = ();
Ouvrir(fichier_de_logs);
ligne_courante = Lire_Ligne(fichier_de_logs);
page = Extraire_Page(ligne_courante);
tant que ! Fin(page) faire
    plus_petit_rang = ∞;
    visite_de_plus_petit_rang = 0;
    pour chaque visite ∈ Visites faire
        numéro_visite = Index(visite, Visites);
        i = 1;
        tant que i ≤ |visite| faire
            si Liées(visite[numéro_visite][i],page) alors
                si plus_petit_rang > i alors
                    plus_petit_rang = i ;
                    visite_de_plus_petit_rang = numéro_visite ;
                    break ;
                fin
            fin
            i ++;
        fin
    fin
    si plus_petit_rang < ∞ alors
        /* Page source trouvée, ajout des données manquantes (le cas
           échéant) à la session la plus probable                        */
        Ajouter_Données_Manquantes(visite_de_plus_petit_rang, plus_petit_rang);
        Ajouter_Donnée(visite_de_plus_petit_rang,page);
    sinon
        /* Aucune page source trouvée, ouverture d'une nouvelle
           session                                                        */
        numéro_visite = Ouvrir_Nouvelle_Visite();
        Ajouter_Donnée(numéro_visite,page);
    fin
    ligne_courante = Lire_Ligne(fichier_de_logs);
    page = Extraire_Page(ligne_courante);
fin
Fermer(fichier_de_logs);
retourner Visites;
```

inaccessible depuis toutes les autres pages vues, il s'agit probablement d'une nouvelle navigation sur le site.

3.3.3 Expérimentations ⋆

Une telle reconstruction, si elle est intuitive, doit être validée. L'idéal serait de comparer un jeu de fichier de logs complets (c.-à.d. sans qu'aucune des requêtes ne soit passée par une machine intermédiaire ni mise en cache), avec un jeu correspondant aux mêmes requêtes dans un régime normal (avec les proxy et caches actifs). Il est malheureusement très difficile de pouvoir générer un ensemble de logs sans contrainte, excepté en local. En effet, le coût humain d'une telle expérimentation est important : il faut dans ce cas obtenir suffisament de navigations locales, c.-à-d. avoir une équipe de testeurs représentatifs dans leur naviguation. Une autre solution, que nous avons choisie, consiste à travailler sur des données artificielles que nous avons générées. Le contrôle sur ses « logs » est donc total, ce qui nous permet d'évaluer uniquement la méthode de reconstruction.

Le protocole expérimental [Mur05a, Mur06] est le suivant : depuis l'extraction d'un graphe d'un site web connu [Eur02], nous générons des navigations utilisateurs artificielles. Pour cela, nous sélectionnons s un nœud du graphe correspond à une page initiale de toutes les navigations générées. Ensuite, nous prenons aléatoirement une série de nœuds $\{p_1,p_2,\ldots\}$ et nous calculons les plus courts chemin de s aux p_i. Ces navigations ne reflétant pas les usages des personnes réelles, nous les adaptons avec une probabilité de ne pas prendre la bonne direction ou de faire un retour arrière. Nous obtenons ainsi des visites constituées de séquences de pages. Ces visites nous servent de référence pour l'évaluation de la méthode de reconstruction. Sur ces données, nous simulons l'utilisation de proxy et de caches pour obtenir un fichier de logs incomplet (celui que nous aurions dans le cas général sur un vrai serveur). De là, nous générons deux jeux de visites à comparer :

- premièrement, nous générons les visites en ne tenant compte que de l'adresse de provenance et du temps (la plupart des produits commerciaux utilisent 30 minutes comme temps inter-visites, nous utilisons 25,5, nombre optimal empiriquement défini dans [CP95][3]) ;
- deuxièmement, nous générons les visites avec notre méthode complète de reconstruction en prenant en compte aussi les paramètres de durée et de provenance.

Lors de l'analyse de logs détériorés par une simulation de proxy-cache, la détection du nombre de visites, dans le premier cas de logs générés, est moindre : de manière évidente, les différences de provenance ou changement de clients sont moins fréquentes. De plus, les pertes de requêtes ainsi que les adresses de provenance erronées induisent des visites détectées sensiblement différentes de celles originales.

Ainsi, pour évaluer la qualité de l'ajout d'information par la reconstruction présentée précédemment, nous définissons deux critères : la différence du nombre de visites

[3] La différence entre 25,5 et 30 minutes influe peu sur les résultats et nous n'avons trouvé aucune étude plus récente pour fixer ce paramètre

détectées et, à l'intérieur des visites mêmes, une distance d'édition [Lev66] sur la séquence des pages les composant. Les résultats sont présentés en figure 3.2.

Les expérimentations ont été menées sur six ensembles de logs artificiels différents.

Les deux indicateurs de qualité se comportent bien : en effet, le nombre de visites détectées est plus proche dans la partie reconstruite et la distance d'édition entre les visites prises deux à deux est aussi plus faible. Ceci veut dire que la reconstruction permet de détecter plus d'incohérences de navigations, et donc plus de visites, mais aussi des visites structurellement plus proches des originales.

Les résultats présentés sont simulés sur 10 machines dont 2 derrière le même proxy. En faisant augmenter ce dernier nombre, le gain en distance devient plus faible mais reste strictement positif. Dans le même temps, le gain sur le nombre de visites décroît, phénomène qui se poursuit toutefois beaucoup moins rapidement.

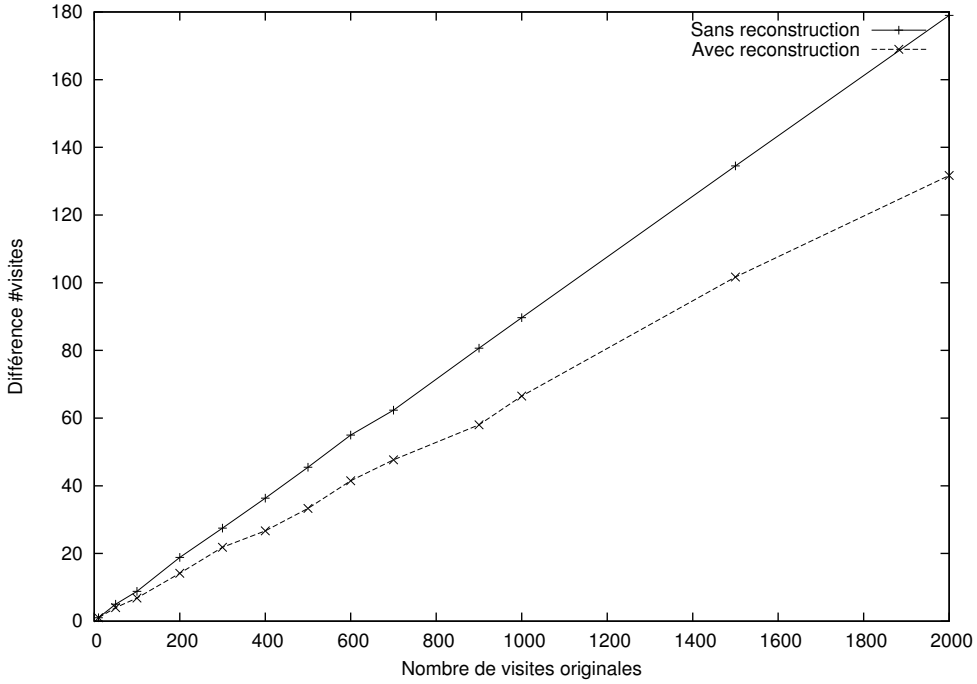
3.3.4 Vers les séquences ⋆

Les visites ainsi détectées sont composées d'une suite de pages vues. Les autres champs que nous extrayons sont la durée de la visite, le nombre de pages vues, l'adresse de provenance, ... Les pages de la visite constituent une séquence que nous recodons en terme de lettres d'un alphabet pour pouvoir servir d'entrée aux algorithmes d'apprentissage automatique. Dans les expérimentations données en dernier chapitre de ce mémoire, certaines menées sur des données artificielles utilisent de plus une simplification de l'ensemble des pages en catégories ou couleur. Durant nos travaux, nous avons développé un outil permettant de colorer les pages d'un site web sur des paramètres syntaxiques et morphologiques. Le partitionnement est fait de manière automatique suivant des régles définies par un expert, portant sur la fréquence d'apparition de mots et du nombre de liens présents dans la page. Ceci permet de réduire la taille de l'alphabet sur lequel travailler en se concentrant sur les problèmes en amont constitués par le prétraitement des données. Une fois ces problèmes réglés, nous utilisons une catégorie par page, ainsi la page est considérée comme une « lettre » de la séquence des pages vues.

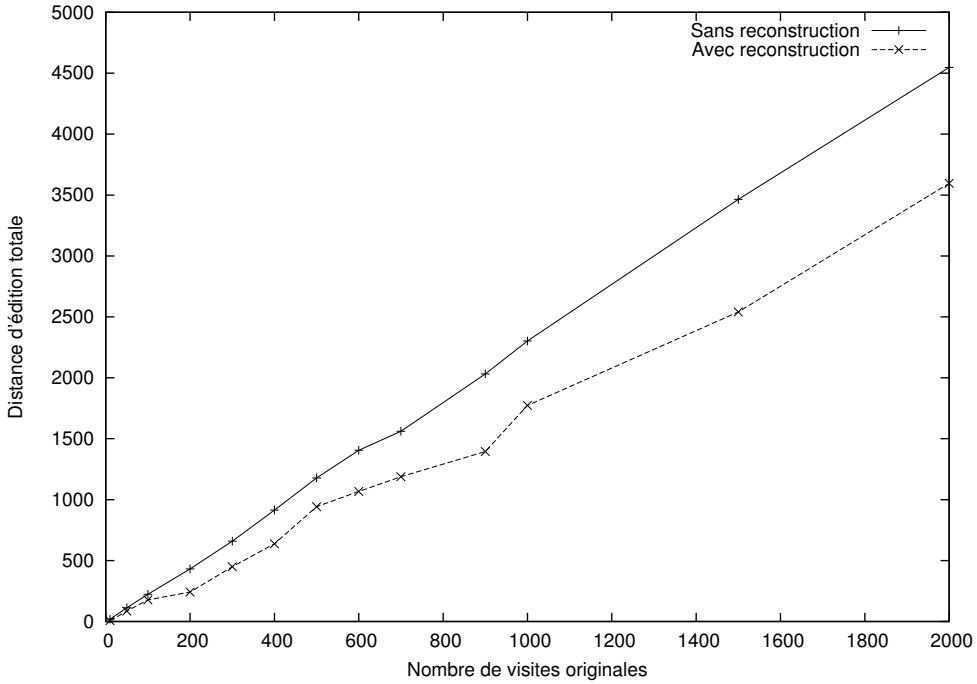
3.4 Conclusion ⋆

Dans ce chapitre, nous avons présenté les fichiers d'enregistrements ou *logs*. Les problèmes de données manquantes et erronées dans ces derniers apparaissent comme rédhibitoires à l'utilisation de l'inférence grammaticale, très sensible aux données bruitées. Nous proposons une méthode de reconstruction des données avec une mise en place d'heuristiques qui donne de bons résultats expérimentaux.

À ce stade, la phase amont du processus d'extraction de connaissance à partir de données se termine. La partie suivante, constituant le cœur du processus, décrit l'apprentissage automatique au moyen de l'inférence grammaticale d'objets stochastiques.



(a) Différence du nombre de visites détectées



(b) Distance d'édition totale

FIG. 3.2 – *Influence de la reconstruction des données*

Deuxième partie

Inférence grammaticale stochastique

Les travaux présentés dans cette partie ont donné lieu à deux communication en conférences internationale et nationale :

[Mdlh04a] T. Murgue et C. de la higuera, *Distances Between Distributions: Comparing Language Models*, Proc. of Syntactical and Structural Pattern Recognition Workshop (Springer-Verlag Berlin Heidelberg 2004, ed.), LNCS, no. 3138, 2004, p. 269–277.

[Mdlh04b] ———, *Distances entre distributions de probabilité : comparaison de modèles de langages stochastiques*, Conférence d'Apprentissage (CAp'04) (Montpellier, France), Presses Universitaires de Grenoble, 2004, p. 97–112.

The idea of a learning machine may appear paradoxical to some readers
A. M. Turing [Tur50]

4 Définitions

Sommaire

Résumé

Le titre de cette nouvelle partie « Inférence grammaticale stochastique » fait référence à l'apprentissage et aux statistiques. Dans la partie précédente, le but était d'obtenir des données sur les utilisateurs les plus fiables possibles. Ces dernières, bien qu'améliorées par la phase de pré-traitement, sont assez volumineuses et souvent bruitées. Pour apprendre avec des données positives seules et souvent imparfaites, nous proposons d'apprendre des modèles structurels stochastiques plus robustes aux bruits.

Nous définissons dans ce chapitre les concepts et représentations que nous utiliserons tout au long de cette partie. La plupart des éléments définis sont des extensions au domaine stochastique d'objets utilisés en inférence grammaticale « classique ». Les modèles stochastiques peuvent être comparés suivant des mesures de similarité.

4.1 Introduction ⋆

Le processus d'extraction de connaissance à partir de données, présenté dans l'introduction, comporte une étape d'apprentissage appelée aussi « fouille » (ou *Mining*). Cette étape importante est souvent considérée comme la seule de tout le processus. Mais nous avons vu qu'il n'en était rien. En effet, c'est du traitement préalable des données que va être établie la qualité de la fouille. Ce prétraitement nous laisse des séquences représentant des navigations d'utilisateurs. Il nous appartient d'en extraire des modèles de navigation.

L'*inférence grammaticale* traite des problèmes d'apprentissage de langages à partir d'un ensemble fini de données. Ces dernières doivent être des mots du langage à apprendre, et dans le cas de présentation complète des mots n'appartenant pas au langage cible. En ce qui nous concerne, les séquences de navigation des utilisateurs sont des mots du langage à inférer, et nous ne disposons par de contre-exemples : toute navigation est valide.

Le processus d'inférence grammaticale généralise peu à peu un ensemble d'exemples pour obtenir un langage défini. Cette induction exacte de langages réguliers (cf. section 4.3) ne peut se faire sans être guidée par des mots hors du langage cible : en effet dans ce cas, le processus ne sait pas quand arrêter la généralisation. Or dans le contexte qui nous intéresse, seules les données des utilisateurs (qui nous serviront d'exemples) sont disponibles. Nous considérons donc par la suite l'apprentissage de langages stochastiques (qui peuvent être appris sous certaines conditions qu'avec des exemples) pouvant être engendrés par une grammaire formelle stochastique.

Avant d'apprendre de tels modèles, nous devons dans ce chapitre définir les notions et notations utilisées dans la suite. Nous présentons donc les objets de base des langages : alphabet, lettre, mot, *etc*. Nous classons les langages suivant leur complexité propre, puis nous définissons les langages stochastiques. Enfin, après avoir défini ces modèles statistiques, nous introduisons des mesures de comparaison entre deux modèles.

4.2 Notions et notations préliminaires

Avant même de parler de langage, nous devons définir les objets de base. Nous invitons le lecteur confirmé à passer cette section.

Le terme « langage », dans la vie de tous les jours, est déjà assez bien défini. Il fait référence aux règles grammaticales à employer pour un texte pour être compris par d'autres personnes. Les concepts triviaux de lettres, mots sont définis de manière formelle dans la suite.

Définition 4.1 (Alphabet) *Nous appelons* $\Sigma = \{a,b,c,d,\dots\}$ *un* alphabet *de taille finie ; c'est un ensemble des lettres* $a,b,c,d\dots$ *Nous notons* $|\Sigma|$ *la taille de l'alphabet* Σ.

Remarque : $\Sigma = \{a,b\}$, est un alphabet fini de taille 2, composé des deux lettres a et b.

Définition 4.2 (Mot et Langage) *Soit un alphabet fini* Σ*, nous appelons indifféremment* mot, séquence *ou* chaîne *une suite finie de lettres appartenant à* Σ*. Nous notons* $u = (u_1,u_2,\dots,u_n)$ *ou plus simplement* $u = u_1u_2\dots u_n$ *le mot de taille* $|u| = n$ *défini sur l'alphabet* Σ*. Le mot vide (correspondant à la suite vide de lettre) sera noté* λ*. Nous notons* Σ^n *l'ensemble de mots de taille* n *construits sur* Σ*,* $\Sigma^{\leq n}$ *l'ensemble de mots de taille au plus* n*. Nous notons* Σ^**, l'ensemble des mots de taille quelconque construits sur l'alphabet* Σ.

Nous notons L *un* langage, *un ensemble de mots :* $L \subseteq \Sigma^*$.

Remarque : Soit $\Sigma = \{a,b\}$, nous dirons que $u = aabba$ est un mot de taille $|u| = 5$ construit sur l'alphabet Σ. Nous dirons de la même manière que $v = aaa = a^3$ est un mot de taille 3 ; cet élément appartient à Σ^* ; il appartient aussi à $\Sigma^{\leq n}$ avec $n \geq 3$.

Nous dirons que $L = \{aabba,aaa\}$ est un langage fini de taille 2.

Définition 4.3 (Concaténation) *La concaténation (.) de deux mots* $u = (u_1,\ldots,u_n)$ *et* $v = (v_1,\ldots,v_k)$ *est* $w = u.v = uv = (u_1,\ldots,u_n,v_1,\ldots,v_k)$. *Cette opération est associative.*

Remarque : Soit Σ un alphabet fini, Σ^* muni de la concaténation (.) — c'est une loi de composition interne — est par définition un *monoïde libre*. λ est l'élément neutre de la loi. Le monoïde est dit régulier car $\forall u,v,w \in \Sigma^*$, $u.v = u.w \Longrightarrow v = w$.

De manière pratique, nous étendons la concaténation aux langages. Soient E,F deux langage de mots, c.-à-d. $E \subseteq \Sigma^*$, $F \subseteq \Sigma^*$, nous notons la concaténation des ensembles par le langage $G = E.F = \{w \in \Sigma^* | \exists u \in E, v \in F \text{ et } w = u.v\}$.

Définition 4.4 (Préfixe, suffixe) *Soit* Σ *un alphabet fini,* $u \in \Sigma^*$, *nous définissons* $v \in \Sigma^*$ *(respectivement* $w \in \Sigma^*$*) comme* préfixe *(resp.* suffixe*) de* u, *s'il existe* $x \in \Sigma^*$ *tel que* $u = vx$ *(resp.* $u = xw$*).*

Remarque : Par exemple, soit $\Sigma = \{a,b\}$, nous dirons que $v = ab$ est un préfixe de $u = ababb$ et nous noterons $\mathrm{Pr}(u) = \{\lambda,a,ab,aba,abab,ababb\}$ l'ensemble des préfixes de u. De même nous noterons $\mathrm{Suf}(u)$ l'ensemble des suffixes du mot u.

Ces définitions primaires nous permettent de définir des objets structurés sur les mots : les langages.

4.3 Langages réguliers stochastiques

Avant de définir formellement un langage stochastique, nous allons définir un langage « classique ». Les grammaires formelles sont une méthode de représentation des langages en tant que systèmes de réécriture, c.-à-d. qu'elles définissent un mot appartenant au langage à partir d'un axiome, et par réécritures successives de ce dernier en suivant des règles grammaticales purement syntaxiques.

Dans [Cho57], CHOMSKY cherchait à formaliser une structure commune à toutes les langues naturelles. De ses travaux, il a défini une hiérarchie de grammaires, spécifiant la difficulté intrinsèque d'une grammaire par rapport aux types de règles de réécriture qu'elle pouvait contenir.

Définition 4.5 (Grammaire) *Une* grammaire *définie sur un alphabet fini* Σ *est un quadruplet* (T,N,R,S) *défini ainsi :*

– T *est l'ensemble des symboles terminaux (*$\Sigma \cup \{\lambda\}$*)*

- *N est l'ensemble des symboles non-terminaux (ou variables)*
- *R est un ensemble de règles de réécriture de la forme $\alpha \mapsto \beta$*
- *S est l'axiome ($S \in N$)*

Les contraintes sur le type de règle définit le type de grammaire :

type 0 *n'impose aucune contrainte sur les règles (c.-à-d. α,$\beta \in (N \cup T)^*$) ; ce sont les grammaires générales.*

type 1 *impose que α appartienne à $T^*.N.T^*$; ce sont les grammaires* sensibles au contexte *ou* sous contexte.

type 2 *impose que α appartienne à N ; ce sont les grammaires* hors contexte.

type 3 *impose que α appartienne à N et que $\beta \in (T^*.N) \cup T^*$; ce sont les grammaires dites* régulières.

Remarque : L'exemple de la table 4.1(a) montre une grammaire formelle très simple ; elle est régulière, elle peut générer tous les mots ayant un nombre quelconque de a, même 0. Ce langage est noté a^*. L'exemple de la table 4.1(b) montre une grammaire formelle hors-contexte, elle peut générer tous les mots ayant le même nombre de a que de b. Ce langage est noté $a^n b^n$.

$$\begin{array}{lcll} T & : & \{a,\lambda\} & \\ N & : & \{S\} & \\ R & : & S & \rightarrow \quad \lambda \\ & & S & \rightarrow \quad aS \end{array}$$

(a) a^*

$$\begin{array}{lcll} T & : & \{a,b,\lambda\} & \\ N & : & \{S\} & \\ R & : & S & \rightarrow \quad \lambda \\ & & S & \rightarrow \quad aSb \end{array}$$

(b) $a^n b^n$

TAB. 4.1 – *Exemple de grammaires formelles*

Un *langage* formel est donc l'ensemble des mots pouvant être générés depuis un axiome en suivant les règles de la grammaire. Ainsi, à chaque type de grammaire est associé un type de langage : les langages généraux, sensibles au contexte, hors-contexte et réguliers. Ces types de grammaires (ou langages) forment une hiérarchie : tout langage de type i est aussi de type $i+1$ ($i \in \{0,1,2\}$).

Définition 4.6 (Langage quotient) *Soit Σ un alphabet fini, L un langage (c.-à-d. $L \subseteq \Sigma^*$), $u \in \Sigma^*$, nous notons* langage quotient *droit de L par u ($L_{/u}$) l'ensemble des mots de Σ^* qui, lorsqu'ils sont concaténés à u, appartiennent à L, soit $\{w \in \Sigma^* | \, uw \in L\}$.*

Remarque : Par exemple, $L = \{aaab,baa,aabb,aa,abaa\}$ et $u = aa$; nous définissons $L_{/u} = \{ab,bb,\lambda\}$.

Dans la suite, nous nous restreignons à l'apprentissage de langages réguliers. Bien qu'étant la classe de langages la plus « simple » de la hiérarchie, ils existent suivant les

cadres d'apprentissage (cf. section 5.3) de nombreux résultats négatifs sur l'apprenabilité des langages de cette classe.

La partie structurelle des modèles permet d'obtenir une grande intelligibilité dans la compréhension humaine du modèle. La représentation graphique des automates accentue encore ce dernier point. L'aspect que nous apportons maintenant est un concept statistique. L'intérêt de considérer des modèles statistiques permet tout d'abord de se positionner dans un cadre d'apprentissage possible (nous verrons que l'apprentissage de modèles exacts sans contre-exemple est impossible), puis lors de la phase d'apprentissage d'être efficace et robuste aux bruits de données.

Nous définissons formellement ces concepts statistiques puis en donnons une représentation graphique : les automates.

Définition 4.7 (Distribution sur les mots) *Soit Σ un alphabet fini, nous définissons une* distribution $\mathfrak{D}$ *sur l'ensemble Σ^* des mots, comme une fonction de probabilité $\mathrm{P}_{\mathfrak{D}} : \Sigma^* \mapsto [0;1]$ telle que $\sum_{w \in \Sigma^*} \mathrm{P}_{\mathfrak{D}}(w) = 1$.*

Définition 4.8 (Langage stochastique) *Soit Σ un alphabet fini, $\mathfrak{D}$ une distribution de probabilité sur l'ensemble Σ^* des mots, un* langage stochastique *L est un ensemble (souvent infini) de couples $(w, \mathrm{P}_{\mathfrak{D}}(w))$ avec $w \in \Sigma^*$.*

L'inférence grammaticale stochastique est le domaine dans lequel nous apprenons un langage stochastique représenté par une grammaire formelle stochastique ou de manière équivalente par un automate déterministe et probabiliste. L'extension des grammaires classiques au cas stochastique est trivial, il suffit d'associer à chaque règle de la grammaire une probabilité d'être utilisée pendant la génération d'un mot. Nous définissons ensuite les modèles stochastiques pour représenter ces langages.

4.4 Automates

Les automates sont de objets mathématiques ayant une représentation graphique sous-jacente simple. Ceci permet une grande compréhension des modèles représentés sous cette forme.

Définition 4.9 (Automate fini déterministe stochastique) *Un* DPFA[1] *ou* automate fini déterministe stochastique *est un quintuplet $\mathcal{A} = (\Sigma_{\mathcal{A}}, Q_{\mathcal{A}}, \delta^{\bullet}_{\mathcal{A}}, \mathrm{p}^{\bullet}_{\mathcal{A}}, \mathrm{f}_{\mathcal{A}})$ où :*

- *$\Sigma_{\mathcal{A}}$ représente un alphabet fini de lettres ;*
- *$Q_{\mathcal{A}}$ est un ensemble fini d'états, $\{q_0, q_1, \ldots, q_{|Q_{\mathcal{A}}|-1}\}$, $q_0 \in Q_{\mathcal{A}}$ est l'état* initial *;*
- *$\delta^{\bullet}_{\mathcal{A}} : Q_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \mapsto Q_{\mathcal{A}}$ définit une fonction de transition, le déterminisme de l'automate est assuré par la condition suivante : $\forall q \in Q_{\mathcal{A}}$, $\forall a \in \Sigma$, $|\{q' \in Q_{\mathcal{A}} : \delta^{\bullet}_{\mathcal{A}}(q,a) = q'\}| \leq 1$*

[1] Acronyme de *Deterministic Probabilistic Finite Automaton*

– $\mathrm{p}^{\bullet}_{\mathcal{A}} : Q_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \mapsto [0,1]$ *est la fonction de probabilité de transition ;*
– $\mathrm{f}_{\mathcal{A}} : Q_{\mathcal{A}} \mapsto [0,1]$ *représente la probabilité pour un état d'être* final.

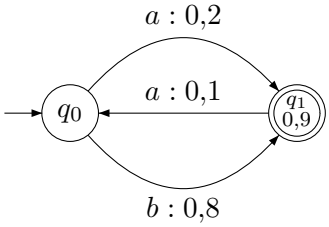


FIG. 4.1 – *Exemple de* DPFA *: $\mathcal{A}$*

Remarque : Considérons, par exemple, l'automate $\mathcal{A}$ de la figure 4.1. Un automate a donc une structure sous-jacente de graphe. Nous représentons les états par les noeuds du graphe (les cercles), et les transitions par les arcs. À chaque arc est liée une probabilité représentant la fonction de probabilité de transition $\mathrm{p}^{\bullet}_{\mathcal{A}}$, à chaque noeud la probabilité d'être final $\mathrm{f}_{\mathcal{A}}$.

Cet automate est composé de $|Q_{\mathcal{A}}| = 2$ états q_0 et q_1, q_0 est l'état initial (flèche entrante), q_1 a une probabilité d'être final non nulle (double cercle), elle vaut $\mathrm{f}_{\mathcal{A}}(q_1) = 0{,}9$, la transition qui lie q_0 à q_1 par la lettre a a une probabilité $\mathrm{p}^{\bullet}_{\mathcal{A}}(q_0,a) = 0{,}2$.

Pour alléger les notations, seules les transitions ayant une probabilité non nulle sont représentées. Mais il faut garder à l'esprit que, dans cet exemple, $\mathrm{p}^{\bullet}_{\mathcal{A}}(q_1,b)$ existe et vaut 0. De même les états ayant un simple cercle comme représentation, ont simplement une probabilité d'être final, nulle.

Sous certaines conditions structurelles et statistiques, un DPFA représente une distribution de probabilité sur l'ensemble des mots (donc un langage stochastique) ; nous voyons ici quelques définitions supplémentaires pour permettre d'établir cette propriété.

Définition 4.10 (Accessibilité) *Soit $\mathcal{A} = (\Sigma_{\mathcal{A}}, Q_{\mathcal{A}}, \delta^{\bullet}_{\mathcal{A}}, \mathrm{p}^{\bullet}_{\mathcal{A}}, \mathrm{f}_{\mathcal{A}})$ un* DPFA*, un état $q \in Q_{\mathcal{A}}$ est dit* accessible *si et seulement s'il existe une suite finie d'états de $Q_{\mathcal{A}}$, $(q_{s_0}, q_{s_1}, \ldots, q_{s_n})$ telle que $q_{s_0} = q_0, q_{s_n} = q$ et $\forall i < n, \exists a \in \Sigma_{\mathcal{A}} |\ \delta^{\bullet}_{\mathcal{A}}(q_{s_i}, a) = q_{s_{i+1}}$.*

Remarque : Ceci peut se réécrire ainsi : il existe un chemin (suite d'arcs) depuis l'état initial vers tout état accessible d'un automate.

Définition 4.11 (Co-accessibilité) *Soit $\mathcal{A} = (\Sigma_{\mathcal{A}}, Q_{\mathcal{A}}, \delta^{\bullet}_{\mathcal{A}}, \mathrm{p}^{\bullet}_{\mathcal{A}}, \mathrm{f}_{\mathcal{A}})$ un* DPFA*, un état $q \in Q_{\mathcal{A}}$ est dit* co-accessible *si et seulement si le fait qu'il soit accessible implique qu'il existe une suite finie d'états de $Q_{\mathcal{A}}$ $(q_{s_0}, q_{s_1}, \ldots, q_{s_n})$ telle que $q_{s_0} = q$, $\mathrm{f}_{\mathcal{A}}(q_{s_n}) > 0$ et $\forall i < n, \exists a \in \Sigma_{\mathcal{A}} |\ \delta^{\bullet}_{\mathcal{A}}(q_{s_i}, a) = q_{s_{i+1}}$.*

Remarque : Ceci peut se réécrire ainsi : il existe un chemin depuis tout état accessible et co-accessible vers un état final.

Pour travailler directement avec les chaînes, plutôt que lettre par lettre, nous généralisons les fonctions de probabilité et de transition associées à un automate ainsi défini.

Définition 4.12 (Fonction de probabilité et de transition étendues) *Soit* $\mathcal{A} = (\Sigma_{\mathcal{A}}, Q_{\mathcal{A}}, \delta^{\bullet}_{\mathcal{A}}, \mathrm{p}^{\bullet}_{\mathcal{A}}, \mathrm{f}_{\mathcal{A}})$ *un* DPFA*, nous appelons fonction de probabilité étendue,* $\mathrm{p}_{\mathcal{A}} : Q_{\mathcal{A}} \times \Sigma^*_{\mathcal{A}} \mapsto [0,1]$ *définie pour* $q \in \Sigma_{\mathcal{A}}, w \in \Sigma^*$ *:*

$$\mathrm{p}_{\mathcal{A}}(q,w) = \begin{cases} \mathrm{f}_{\mathcal{A}}(q) & \text{si } w = \lambda \\ \mathrm{p}^{\bullet}_{\mathcal{A}}(q,a).\, \mathrm{p}_{\mathcal{A}}\big(\delta^{\bullet}_{\mathcal{A}}(q,a),x\big) & \text{sinon. } (w = ax, a \in \Sigma, x \in \Sigma^*) \end{cases}$$

Nous appelons fonction de transition étendue, $\delta_{\mathcal{A}} : Q_{\mathcal{A}} \times \Sigma^*_{\mathcal{A}} \mapsto Q_{\mathcal{A}}$ *définie pour* $q \in \Sigma_{\mathcal{A}}, w \in \Sigma^*$ *:*

$$\delta_{\mathcal{A}}(q,w) = \begin{cases} q & \text{si } w = \lambda \\ \delta_{\mathcal{A}}\big(\delta^{\bullet}(q,a),x\big) & \text{sinon. } (w = ax, a \in \Sigma, x \in \Sigma^*) \end{cases}$$

Remarque : Dans la suite, lorsqu'aucune confusion n'est possible, nous ne noterons volontairement pas l'indice $\mathcal{A}$, sur les constituants de l'automate ni leur extension.

Le théorème suivant permet la représentation d'un langage stochastique par un DPFA, en effet il définit les conditions nécessaires et suffisantes pour qu'un automate de ce type représente de manière équivalente une distribution de probabilités sur l'ensemble des chaînes de l'alphabet.

Théorème 4.1 (Équivalence entre automate et distribution de probabilité) *Soit* $\mathcal{A}$ *un* DPFA*, il est équivalent à une distribution de probabilité sur les mots de* Σ *si et seulement si :*

1. $\forall q \in Q$*, q est co-accessible ;*
2. $\forall q \in Q$ *accessible,* $\mathrm{f}_{\mathcal{A}}(q) + \sum_{a \in \Sigma} \mathrm{p}^{\bullet}_{\mathcal{A}}(q,a) = 1$

Preuve : La preuve bien que technique n'est pas compliquée. Elle est décrite dans [Tho00]. Cette dernière mentionne également que chaque état de l'automate pris comme état initial définit une distribution de probabilité sur Σ^*, c.-à-d. $\forall q \in Q$, $\sum_{w \in \Sigma^*} \mathrm{p}_{\mathcal{A}}(q,w) = 1$. □

Remarque : Un DPFA $\mathcal{A}$ définit une distribution $\mathfrak{D}$ sur Σ^* : soit $w \in \Sigma^*, \mathrm{P}_{\mathfrak{D}}(w) = \mathrm{P}_{\mathcal{A}}(w) = \mathrm{p}_{\mathcal{A}}(q_0,w)$.

Ainsi la probabilité d'un mot construit sur l'alphabet peut se calculer de proche en proche, par multiplication successive directement *via* l'automate. En considérant l'automate de la figure 4.1, nous présentons dans les tables 4.2 le détail du calcul des probabilités des mots aaa et $aaba$.

Nous dirons donc que aaa *appartient* au langage engendré par l'automate alors que $aaba$ *n'appartient pas* à ce langage.

$$
\begin{aligned}
&\mathrm{P}_{\mathfrak{D}}(aaa) = \mathrm{p}(q_0,aaa)\\
&\mathrm{P}_{\mathfrak{D}}(aaa) = \mathrm{p}^{\bullet}(q_0,a) \times \mathrm{p}(q_1,aa)\\
&\mathrm{P}_{\mathfrak{D}}(aaa) = 0{,}2 \times \mathrm{p}^{\bullet}(q_1,a) \times \mathrm{p}(q_0,a)\\
&\mathrm{P}_{\mathfrak{D}}(aaa) = 0{,}2 \times 0{,}1 \times \mathrm{p}^{\bullet}(q_0,a) \times \mathrm{p}(q_1,\lambda)\\
&\mathrm{P}_{\mathfrak{D}}(aaa) = 0{,}2 \times 0{,}1 \times 0{,}2 \times \mathrm{f}(q_1)\\
&\mathrm{P}_{\mathfrak{D}}(aaa) = 0{,}2 \times 0{,}1 \times 0{,}2 \times 0{,}9 = 0{,}0036
\end{aligned}
$$

(a) *aaa*

$$
\begin{aligned}
&\mathrm{P}_{\mathfrak{D}}(aaba) = \mathrm{p}(q_0,aaba)\\
&\mathrm{P}_{\mathfrak{D}}(aaba) = \mathrm{p}^{\bullet}(q_0,a) \times \mathrm{p}(q_1,aba)\\
&\mathrm{P}_{\mathfrak{D}}(aaba) = 0{,}2 \times \mathrm{p}^{\bullet}(q_1,a) \times \mathrm{p}(q_0,ba)\\
&\mathrm{P}_{\mathfrak{D}}(aaba) = 0{,}2 \times 0{,}1 \times \mathrm{p}^{\bullet}(q_0,b) \times \mathrm{p}(q_1,a)\\
&\mathrm{P}_{\mathfrak{D}}(aaba) = 0{,}2 \times 0{,}1 \times 0{,}8 \times \mathrm{p}^{\bullet}(q_1,a) \times \mathrm{p}(q_0,\lambda)\\
&\mathrm{P}_{\mathfrak{D}}(aaba) = 0{,}2 \times 0{,}1 \times 0{,}2 \times 0{,}8 \times 0{,}2 \times \mathrm{f}(q_0)\\
&\mathrm{P}_{\mathfrak{D}}(aaba) = 0{,}2 \times 0{,}1 \times 0{,}2 \times 0{,}8 \times 0{,}2 \times 0 = 0
\end{aligned}
$$

(b) *aaba*

Tab. 4.2 – *Exemples de calcul de probabilités sur les mots*

Nous définissons pour tout mot w construit sur l'alphabet Σ, pour tout $0 \leq i \leq |w|$, $\mathrm{Pr}_i(w)$ comme la sous-chaîne de taille i préfixe du mot w. Par exemple, si $w = abbabba$, $\mathrm{Pr}_3(w) = abb$, $\mathrm{Pr}_0(w) = \lambda$. De plus, nous définissons, pour $1 \leq i \leq |w|$, par le symbolisme $w[i]$ la i^{e} lettre du mot w.

Propriété 4.1

$$\forall w \in \Sigma^*, \mathrm{p}(q,w) = \prod_{i=1}^{|w|} \Big(\mathrm{p}^{\bullet}\big(\delta\big(q, \Pr_{i-1}(w)\big), w[i]\big)\Big) \times \mathrm{f}\big(\delta(q,w)\big)$$

Preuve : Cette propriété est simplement la réécriture itérative de la définition récursive du calcul de la probabilité d'un mot dans un automate. □

Les automates ne possédant que des états *utiles* (accessibles et co-accessibles) sont dits « *émondés* ». Dans la suite nous ne considérons que des automates de ce type.

4.5 Mesures de similarité entre modèles de langages

Lorsque nous disposons de plusieurs modèles de langages construits sur le même alphabet fini Σ, il est important de pouvoir connaître une mesure de similarité entre ces modèles. En effet, lors de l'apprentissage d'un langage stochastique, nous verrons que nous procédons de proche en proche en généralisant le langage en cours d'apprentissage suivant plusieurs critères. Il est donc primordial de savoir si la généralisation ne nous emmène par trop « loin » de la distribution initiale. De plus, en dehors du cadre d'apprentissage ou en terme d'évaluation de celui-ci, il est nécessaire d'avoir des outils de comparaison de modèles de langages, et nécessaire de savoir si un langage est « proche » ou non d'un autre.

Nous allons présenter dans la suite deux mesures de similarité. Certaines distances « classiques » utilisées dans d'autres contextes peuvent être définies dans ce cadre. Le point important des mesures que nous allons présenter est qu'elles sont calculables directement sur les distributions représentées sous forme de DPFA. Nous considérons

par la suite un alphabet fini Σ et deux distributions de probabilité sur les mots $\mathfrak{D}$, $\mathfrak{D}'$, ayant respectivement $\mathrm{P}_{\mathfrak{D}}$ et $\mathrm{P}_{\mathfrak{D}'}$ comme fonctions de probabilité associées.

4.5.1 Divergence de Kullback-Leibler

Cette divergence fait appel à la notion d'entropie ; c'est une notion statistique définissant le degré d'incertitude d'une variable aléatoire, ou le risque empirique lorsqu'elle est calculée sur un ensemble fini de mots. La mesure présentée ci-après est aussi appelée *entropie relative* : elle représente une mesure de similarité entre deux distributions suivant leur entropie.

Définition 4.13 (Divergence de Kullback-Leibler) *La différence d'entropie relative ou* divergence de Kullback-Leibler *de $\mathfrak{D}$ relativement à $\mathfrak{D}'$ est définie [Kul59] ainsi :*

$$\mathrm{d}_{KL}(\mathfrak{D},\mathfrak{D}') = \sum_{w\in\Sigma^*} \mathrm{P}_{\mathfrak{D}}(w) \log \frac{\mathrm{P}_{\mathfrak{D}}(w)}{\mathrm{P}_{\mathfrak{D}'}(w)}$$

Remarque : Pour que cette mesure soit définie presque partout, il est indispensable de définir les valeurs de deux expressions mathématiques : soit $x \in [0,1], 0 \times \log \frac{0}{x} = 0$ et $x \in]0,1], x \times \log \frac{x}{0} = \infty$; ces notions sont couramment adoptées pour des raisons de continuité.

Cette mesure de similarité entre distributions de probabilités n'est pas une distance au sens mathématique du terme. En effet, elle n'est pas symétrique, et ne respecte pas l'inégalité triangulaire (les autres propriétés classiques pour une distance sont, elles, respectées : $\mathrm{d}_{KL}(\mathfrak{D},\mathfrak{D}') = 0 \iff \forall w \in \Sigma^*,\ \mathrm{P}_{\mathfrak{D}}(w) = \mathrm{P}_{\mathfrak{D}'}(w)$, et $\mathrm{d}_{KL}(\mathfrak{D},\mathfrak{D}') \geq 0$, elles proviennent de l'inégalité de Gibb's).

Avec les définitions adoptées, nous pouvons voir que l'entropie relative devient infinie dès que la distribution $\mathfrak{D}'$ donne une probabilité nulle à un mot alors que la distribution $\mathfrak{D}$ donne une probabilité strictement positive. Ce dernier élément est très problématique pour comparer deux langages : en effet, il suffit de trouver un mot appartenant au langage représenté par la première distribution qui n'appartient pas au deuxième langage pour que la divergence soit infinie. Pour éviter ceci, il existe des méthodes de lissage de distributions de probabilité dont la nature est de donner une probabilité strictement positive à tous les mots de Σ^* en redistribuant de manière adéquate (la somme doit toujours être égale à 1) les probabilités sur l'ensemble des mots.

4.5.2 Distance euclidienne ⋆

Nous avons une idée naturelle de ce que représente une distance euclidienne dans l'espace ou dans le plan. Nous vivons dans un espace à trois dimensions[2] et donc l'habitude d'appréhender facilement des notions de distances. La distance euclidienne dans $\mathbb{R}^n$ où n représente le nombre de dimensions de l'espace est vraiment classique,

[2] Au moins en première approximation, certaines théories visant à définir un espace physique de dimension supérieure ou non entière

nous nous intéressons à son extension aux DPFA représentant une distribution de probabilité. Chaque mot représente ainsi une dimension possible de l'espace (qui devient donc de dimension infinie) et un automate est repéré dans cette espace *via* un vecteur infini où chaque composante représente la probabilité du mot attribuée par l'automate.

Considérons Σ^*, nous le munissons de l'ordre lexicographique[3] naturel : cet ordre est total. Nous pouvons ainsi définir un automate par un vecteur de probabilités, celles qu'il affecte à chaque mot. La distance euclidienne peut être étendue à cet espace [MH04b, MH04a].

Définition 4.14 (Distance euclidienne) *Nous définissons la* distance euclidienne *entre ces deux entités comme :*

$$\mathrm{d}_2(\mathfrak{D},\mathfrak{D}') = \sqrt{\sum_{w\in\Sigma^*} \left(\mathrm{P}_{\mathfrak{D}}(w) - \mathrm{P}_{\mathfrak{D}'}(w)\right)^2}$$

Nous montrerons dans les chapitres 7 et 8 que cette dernière mesure est une distance formelle au sens mathématique du terme et comment calculer ces mesures : nous définirons la perplexité (une mesure liée à la d_{KL}) et deux distances, la d_2 et son extension aux préfixes des mots.

4.6 Conclusion ⋆

Dans ce chapitre, nous avons défini les objets dont nous allons avoir besoin dans la suite : les langages réguliers stochastiques et leur représentation par des DPFA. Nous avons aussi présenté rapidement deux mesures de similarité entre distribution de probabilité. Le chapitre 5 définit le principal concept de l'inférence grammaticale : qu'est-ce qu'« apprendre » ?

[3] L'ordre lexicographique est défini ainsi : $(a_1,a_2,a_3,\ldots) <_{\text{lex}} (b_1,b_2,b_3,\ldots) \iff a_1 < b_1$ ou $\big(a_1 = b_1$ et $a_2 < b_2\big)$ ou $\big(a_1 = b_1$ et $a_2 = b_2$ et $a_3 < b_3\big)$ ou $\ldots$

5 Théorie de l'inférence grammaticale stochastique

Sommaire

Résumé

Dans ce chapitre, nous proposons de définir formellement le concept d'apprentissage automatique comme la génération d'un modèle (hypothèse) à partir de données (exemples). Nous présentons des cadres d'apprentissage qui définissent ce qu'est un bon modèle appris en régissant le type d'entrée, le type de méthode d'apprentissage, la classe des objets à apprendre, la représentation utilisée.

5.1 Introduction ⋆

« Apprendre » humainement un concept est une notion cognitive bien maîtrisée mais mettant en jeu d'importants mécanismes intellectuels. De manière formelle, nous verrons qu'apprendre artificiellement lors d'un processus inductif comme celui de l'inférence grammaticale revient à présenter une hypothèse cohérente avec les exemples donnés. L'évaluation de cette hypothèse est ensuite un point majeur pour savoir si l'algorithme a « bien » appris.

Pour qu'un problème d'inférence grammaticale soit défini complètement, nous devons définir :

- la classe des grammaires à apprendre, $\mathcal{C}$;
- le langage de description des objets à apprendre (espace des hypothèses, $\mathcal{H}$) ;
- le type d'exemples en entrée du problème ;
- les critères d'évaluation de l'hypothèse $h \in \mathcal{H}$ proposée (ou cadre d'apprentissage) ;
- la classe d'algorithmes utilisée pour fabriquer l'hypothèse h.

Le problème qui nous importe, à savoir extraire un modèle de comportements d'utilisateur à partir de séquences de navigation, peut donc être défini formellement de multiples façons. Nous voyons dans ce chapitre quels pourraient être les divers choix possibles pour chacun des points définissant le problème et nous argumentons sur les solutions retenues par la suite.

Nous avons fixé dans le chapitre précédent la classe à apprendre ($\mathcal{C}$) aux grammaires stochastiques régulières : en effet nous verrons qu'il existe même sur cette classe la plus « simple » nombre de résultats négatifs d'apprenabilité. De même, nous avons vu que les automates finis déterministes stochastiques (DPFA) étaient, sous certaines conditions structurelles, une représentation équivalente aux grammaires régulières stochastiques. L'ensemble des automates remplissant ces conditions donneront $\mathcal{H}$. Les algorithmes que nous présenterons dans le chapitre suivant seront tous basés sur cette représentation des grammaires.

Dans le cadre d'inférence de grammaires stochastiques régulières nous verrons en quoi le type d'exemples et la façon dont ils sont présentés sont importants. Enfin nous présenterons quelques résultats théoriques sur l'apprenabilité de ces grammaires dans des cadres que nous aurons définis rigoureusement.

5.2 Ensemble d'exemples

Le principe inductif de l'inférence est simple : partir d'un ensemble d'exemples du langage à trouver et le généraliser suffisamment (mais pas trop) pour obtenir ce langage. Les exemples donnés en entrée de l'algorithme vont donc jouer un rôle essentiel dans la recherche de la cible. Dans le cadre classique de l'inférence, les exemples donnés sont soit *positifs* (c'est à dire qu'ils appartiennent au langage), soit *négatifs* (ils n'y appartiennent pas). Dans le cas stochastique, nous verrons les exemples comme des mots ayant une probabilité d'appartenir au langage défini, pouvant être nulle.

Définition 5.1 (Présentation stochastique) *Nous appelons* présentation stochastique *d'une distribution de probabilité $\mathfrak{D}$, une suite infinie de mots appartenant à Σ^* dont la fréquence d'apparition est conforme à des tirages indépendants selon $\mathfrak{D}$.*

Nous noterons S une telle présentation, et par abus de notation S_p les p premiers éléments de celle-ci qui nous serviront d'échantillon.

> *Remarque :* L'échantillon est en fait un *multi-ensemble* : ensemble où une occurrence peut apparaître plusieurs fois. Par exemple, l'échantillon $S_{20} = \{b,b,a,a,b,b,b,b,a,b,b,aab,b,b,b,b,babab,b,a,a\}$ et tiré suivant la distribution représentée par l'automate de la figure 4.1 de la page 52.
>
> Soit X un multi-ensemble, nous définissons par $\mathrm{Occ}(u,X)$ (respectivement $\mathrm{OccP}(u,X)$) le nombre d'occurrences de u dans X (resp. le nombre d'occurrences de w dans X avec $w = uv, v \in \Sigma^*$).

Après avoir fixé ou défini les premiers éléments du problème d'inférence grammaticale stochastique, il nous reste maintenant à définir ce qu'est un « bon » apprentissage.

5.3 Cadres d'apprentissage

Ces cadres d'apprentissage définissent réellement ce que veut dire apprendre. En effet, ils formalisent les critères retenus pour un apprentissage de qualité. Dans le paradigme stochastique de l'inférence grammaticale, deux cadres ont été étudiés : ils sont tous deux une extension d'un cadre existant dans le contexte non statistique. Nous présentons ci-après le modèle PAC et l'identification à la limite.

5.3.1 Cadre PAC

Le modèle PAC[1] a été introduit par Valiant dans [Val84]. Ce cadre définit un bon apprentissage comme un algorithme capable de créer avec une probabilité « assez proche » de 1, une hypothèse « pas trop loin » de l'hypothèse optimale.

Définition 5.2 (PAC-apprenabilité) *Une classe de grammaires $\mathcal{C}$ est* PAC-apprenable *via une représentation par un* DPFA *s'il existe un algorithme qui, pour tout $L \in \mathcal{C}$ défini sur l'alphabet Σ représenté par un automate $\mathcal{A}$ à n états, et, étant donnés un échantillon stochastique S, deux entiers δ et ϵ, retourne une hypothèse $\mathcal{B}$ et vérifie les propriétés suivantes :*

- *le temps de génération de $\mathcal{B}$ est polynomial en $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|\Sigma|$, n, $|S|$;*
- $\mathrm{P}\big(\mathrm{d}_{KL}(\mathcal{A},\mathcal{B}) < \epsilon\big) > 1-\delta$

Remarque : Initialement la partie « approximativement » du cadre est donnée par la divergence de Kullback-Leibler entre la cible et l'hypothèse. En changeant cette mesure de similarité par la distance[2] d_∞ des travaux ont montré qu'il était possible d'apprendre dans ce sens la classe des langages réguliers en renvoyant simplement une représentation de l'apprentissage par cœur des données.

La mesure d'approximation est donc primordiale et la d_{KL} offre un bon compromis entre distinction des modèles et calculabilité.

Dans ce cadre d'apprentissage les résultats d'apprenabilité sont pour l'instant plutôt négatifs. En effet, un résultat de Kearns *et al.* [KMR+94] démontre que sous une conjecture[3] de non-apprenabilité de fonctions de parité bruitées, la classe des DPFA définissant des distributions de probabilité sur des chaînes de taille n n'est pas PAC-apprenable. Les seuls résultats positifs sont donnés sur des sous-classes d'automates : tout d'abord Ron *et al.* [RST95] démontrent que les automates stochastiques sans cycles sont PAC-apprenables, ensuite Clark et Thollard ont montré dans [CT04] que sous des contraintes structurelles (taille de l'automate bornée et taille moyenne des chaînes générées bornée) les DPFA sont PAC-apprenables.

Ce cadre d'apprentissage, suivant la mesure de similarité utilisée, est soit trop permissif soit pas assez. En effet, en considérant la d_∞, l'apprentissage par cœur suffirait à apprendre (alors que ce n'est pas en pratique un bon apprentissage), et en considérant

1 pour *Probably Approximately Correct*

2 $\mathrm{d}_\infty(\mathfrak{D},\mathfrak{D}') = \max_{w\in\Sigma^*}\left|\mathrm{P}_{\mathfrak{D}}(w) - \mathrm{P}_{\mathfrak{D}'}(w)\right|$

3 Noisy Parity Assumption

la d_{KL}, les résultats sur la classe entière ne sont pas connus mais paraissent difficiles au regard de la non-apprenabilité des DPFA définissant des distributions de probabilité sur des chaînes de taille n.

Nous définissons ci-après un autre cadre souvent utilisé, c'est l'identification à la limite : là encore le cadre que nous présentons est une extension au domaine stochastique d'un cadre de l'inférence classique.

5.3.2 Identification à la limite

Ce cadre théorique a été introduit dans sa version stochastique dans [HT00] et dans sa version classique dans les travaux de Gold [Gol78]. Contrairement au cadre précédent, nous voulons dans ce contexte apprendre strictement la distribution, pour cela nous définissons formellement le paradigme.

Définition 5.3 (Identification à la limite avec probabilité 1**)** *La classe de distributions $\mathcal{C}$ est* identifiable à la limite avec probabilité 1 *s'il existe un algorithme qui, pour toute distribution $\mathfrak{D} \in \mathcal{C}$ définie sur l'alphabet Σ, pour toute présentation stochastique de $\mathfrak{D}$ notée S, pour tout $p \in \mathbb{N}$, retourne une hypothèse $h(S_p)$ et vérifie la propriété suivante : $\exists n \in \mathbb{N}, \forall k \in \mathbb{N} > n, h(S_k) = h(S_n) = \mathfrak{D}$.*

Dans ce cadre, de la Higuera *et al.* ont démontré deux résultats importants. Tout d'abord, un résultat positif : dans [HT00], il est démontré que les automates finis déterministes stochastiques ayant une fonction de probabilité $\mathrm{p}^\bullet$ à valeurs dans $\mathbb{Q}$ étaient identifiables à la limite avec probabilité 1. À l'opposé, dans [HO04], l'identification des DPFA est mise en défaut dans le cas où l'on veut un échantillon d'entrée de taille polynomiale par rapport à la représentation sous forme d'automate de la cible.

5.4 Conclusion ⋆

Dans ce chapitre nous avons présenté le principe d'apprentissage de manière formelle, dans la suite nous présenterons des algorithmes cohérents avec le cadre « identification à la limite avec probabilité 1 » retournant comme hypothèses des DPFA à valeurs dans $\mathbb{Q}$.

De nombreux travaux ont été menés pour mettre en place des algorithmes de ce type. Diverses méthodes globales ont été avancées : sur les modèles de Markov cachés (structures équivalentes [DDE05] sous certaines conditions aux DPFA), plusieurs personnes se sont essayées à la fission d'états, la suppression de transitions ou la correction d'erreurs. Mais la grande majorité des travaux utilisent des algorithmes généralisant l'échantillon d'entrée par fusions d'états. Nous présentons cela dans le chapitre suivant.

6 Algorithmes d'inférence par fusions d'états

Sommaire

Résumé

Dans ce chapitre nous allons voir l'algorithme générique le plus souvent employé en inférence grammaticale probabiliste. Nous verrons qu'il comporte plusieurs étapes : un apprentissage par cœur des données d'apprentissage suivi d'une recherche de la cible par une série de généralisations structurelles et statistiques. Les parties importantes de l'algorithme générique sont étudiées en détails ainsi que deux instanciations ALERGIA et MDI

6.1 Introduction ⋆

Nous avons vu dans le chapitre précédent les formalismes usuels de cadres d'apprentissage. Nous nous plaçons maintenant dans la partie de descriptions d'algorithmes cohérents avec le cadre d'identification à la limite. Nous définissons dans un premier temps un objet de représentation pour un apprentissage par cœur des données. Nous montrons dans la suite que l'apprentissage d'un automate intéressant (à savoir non appris par cœur) peut être vu comme la recherche d'une cible structurelle dans un espace bien défini, puis de l'estimation des probabilités de la structure. En effet, les propriétés présentées ci-après sont souvent des extensions de l'apprentissage de langages non stochastiques dans lesquels la gestion des probabilités a été rajoutée.

Dans ce chapitre nous montrons tout d'abord ce qu'est le point de départ d'un algorithme générique en apprentissage stochastique, puis nous formalisons l'espace de recherche de la cible structurelle. Nous présentons aussi une méthode de généralisation de langage : la fusion d'états. Puis, nous présentons l'algorithme dans son ensemble et nous nous attardons sur les points à définir pour décliner l'algorithme sous plusieurs formes.

En fin de chapitre, nous présentons les deux algorithmes les plus connus de ce domaine, ayant de très bons résultats pratiques sur les données, notamment en langue naturelle.

6.2 Apprentissage par cœur

Nous parlons d'apprentissage ou d'inférence de langages. Le point de départ des méthodes que nous présentons un peu plus loin est une représentation par un automate stochastique de la distribution empirique représentée par l'échantillon de données d'entrée. Cette représentation est un objet appelé PPTA.

Définition 6.1 (PPTA[1]) *Le* PPTA *d'un échantillon stochastique S_p est un* DPFA PPTA_{S_p} *défini comme suit :*

- Σ_{PPTA} *correspond à l'alphabet de l'échantillon S_p ;*
- $Q_{\mathrm{PPTA}} = \bigcup_{x \in S_p} \mathrm{Pr}(x)$, $q_0 = \lambda$;
- $\forall q \in Q_{\mathrm{PPTA}}, \forall a \in \Sigma_{\mathrm{PPTA}}$, $\delta^\bullet(q,a) = q.a$;
- $\forall q \in Q_{\mathrm{PPTA}}, \forall a \in \Sigma_{\mathrm{PPTA}}$, $\mathrm{p}^\bullet(q,a) = \frac{\mathrm{OccP}(q.a,S_p)}{\mathrm{OccP}(q,S_p)}$;
- $\forall q \in Q_{\mathrm{PPTA}}$, $\mathrm{f}(q) = \frac{\mathrm{Occ}(q,S_p)}{\mathrm{OccP}(q,S_p)}$.

Remarque : Par construction le PPTA d'un échantillon de mots est un DPFA dont les probabilités sont définies dans $\mathbb{Q}$. Pour pouvoir calculer facilement les probabilités des transitions des automates quotients (cf. Définition 6.3), nous considérons dans la suite que les probabilités de transitions et d'être final sont dans $\mathbb{Q}$ avec, $\mathrm{p}^\bullet(q,a) = \frac{\mathrm{cl}(q,a)}{\mathrm{ct}(q)}$ et $\mathrm{f}(q) = \frac{\mathrm{cf}(q)}{\mathrm{ct}(q)}$. Nous dirons que $\mathrm{cl}(q,a)$ (resp. $\mathrm{cf}(q)$ et $\mathrm{ct}(q)$) représente le compteur de la lettre a pour l'état q (resp. le compteur de fin de l'état q et le compteur total de l'état q). Ces compteurs sont donc initialisés dans le PPTA ($\mathrm{cl}(q,a) = \mathrm{OccP}(q.a,S_p)$, $ct(q) = \mathrm{OccP}(q,S_p)$ et $\mathrm{cf}(q) = \mathrm{Occ}(q,S_p)$) et vont être modifiés au gré des fusions. Dans le cas où il pourrait y avoir ambiguïté, nous indicerons les différents compteurs par l'automate auquel ils se rapportent.

Ainsi définis, les états du PPTA sont nommés par le préfixe des mots qu'ils représentent ; pour faire correspondre cette définition à celle des DPFA classiques, il suffit de renommer chaque état en q_i avec i représentant le rang du préfixe dans S_p muni de l'ordre hiérarchique[2]. Ainsi l'état de λ devient q_0, celui de a devient q_1, ...

1 De l'anglais *Probabilistic Prefix Tree Acceptor = arbre accepteur de préfixes probabiliste*

2 L'ordre hiérarchique est défini ainsi : $a <_{\mathrm{h}} b \iff |a| < |b|$ ou $\big(|a| = |b|$ et $a <_{\mathrm{lex}} b\big)$

L'automate de la figure 6.1 est le PPTA de l'ensemble S_{20} de la définition 5.1 de la page 58. Par souci de clarté, les états munis d'une probabilité d'être finale nulle sont représentés sans cette valeur.

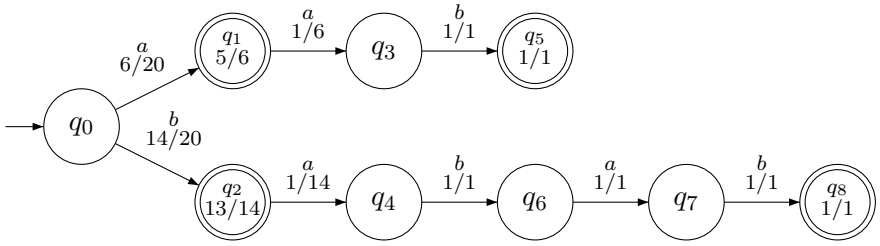


FIG. 6.1 – *Exemple de* PPTA

Ce type d'automate représente donc exactement les données d'entrée. Cet apprentissage par cœur n'est pas intéressant pour nous. En effet, nous désirons apprendre un modèle reconnaissant les données d'entrée mais aussi ayant la capacité de reconnaître des mots du langage ne faisant pas partie de l'échantillon d'apprentissage. Pour cela, il faut généraliser le langage : une méthode couramment utilisée est la fusion d'états dans un DPFA.

6.3 Fusion et déterminisation

Nous avons besoin de définir les opérations de base sur les états des automates pour définir formellement une fusion d'états. Le principe est simple : deux états sont sélectionnés et « fusionnés » en un seul. Pour que la structure ainsi obtenue reste un DPFA représentant une distribution sur les mots, quelques manipulations restent à faire.

Nous nous inspirons ci-après des notations définies dans [DM98].

Définition 6.2 (Partition) *Soit un ensemble d'éléments X. Une* partition π *de X est un ensemble de sous-ensembles non vides de X tels que leur union soit X et que les intersections de ses ensembles deux à deux soient vides. Ces sous-ensembles disjoints sont appelés « blocs ».*

Remarque : Soit $X = \{1,2,3\}$, $\pi = \{\{1\},\{2,3\}\}$, $\pi' = \{\{1,2\},\{3\}\}$ sont des partitions de X. Par contre $\{\{1,2\},\{1,3\}\}$ et $\{\{1\},\{3\}\}$ n'en sont pas. Les sous-ensembles de X sont appelés blocs dans une partition, et nous dirons que $\mathrm{B}(1,\pi) = \{1\}$ est l'unique bloc contenant 1 dans la partition π.

Nous définissons maintenant la fusion d'états sur un DPFA, cette opération renvoie un DPFA appelé automate quotient.

Définition 6.3 (Automate quotient) *Soit* $\mathcal{A} = (\Sigma_\mathcal{A}, Q_\mathcal{A}, \delta^\bullet_\mathcal{A}, \mathrm{p}^\bullet_\mathcal{A}, \mathrm{f}_\mathcal{A})$ *un* DPFA[3], *soit* π *une partition de l'ensemble des états de* $\mathcal{A}$, $Q_\mathcal{A}$. *Nous définissons* $\mathcal{B} = \mathcal{A}_{/\pi} = (\Sigma_\mathcal{B}, Q_\mathcal{B}, \delta^\bullet_\mathcal{B}, \mathrm{p}^\bullet_\mathcal{B}, \mathrm{f}_\mathcal{B})$ *tel que*

- $\Sigma_\mathcal{B} = \Sigma_\mathcal{A}$;
- $Q_\mathcal{B} = \{B(q,\pi) | \, q \in Q_\mathcal{A}\}$;
- *la fonction* $\delta^\bullet_\mathcal{B}$ *est définie ainsi :* $\forall B \in Q_\mathcal{B}, \forall a \in \Sigma$, $\delta^\bullet_\mathcal{B}(B,a) = B' \in Q_\mathcal{B} \iff \exists q,q' \in Q_\mathcal{A}$ *tel que* $q \in B, q' \in B'$ *et* $\delta^\bullet_\mathcal{A}(q,a) = q'$;
- *la fonction* $\mathrm{p}^\bullet_\mathcal{B}$ *est définie ainsi :*

$$\forall B \in Q_\mathcal{B}, \forall a \in \Sigma, \ \mathrm{p}^\bullet_\mathcal{B}(B,a) = \frac{\sum_{q \in Q_\mathcal{A} \cap B} \mathrm{cl}_\mathcal{A}(q,a)}{\sum_{q \in Q_\mathcal{A} \cap B} \mathrm{ct}_\mathcal{A}(q)}$$

- *la fonction* $\mathrm{f}_\mathcal{B}$ *est définie ainsi :*

$$\forall B \in Q_\mathcal{B}, \ \mathrm{f}_\mathcal{B}(B) = \frac{\sum_{q \in Q_\mathcal{A} \cap B} \mathrm{cf}_\mathcal{A}(q)}{\sum_{q \in Q_\mathcal{A} \cap B} \mathrm{ct}_\mathcal{A}(q)}$$

Remarque : Le langage représenté par un automate $\mathcal{A}$ est toujours inclus dans $\mathcal{A}_{/\pi}$,$\forall \pi$ partition des états de $\mathcal{A}$.

La figure 6.2 montre un exemple d'automate quotient : dans ce cas précis, $\mathcal{A}$ est le PPTA de la figure 6.1 et $\pi = \{\{q_0\},\{q_1\},\{q_2\},\{q_3,q_4\},\{q_5\},\{q_6\},\{q_7\},\{q_8\}\}$. Dans cet

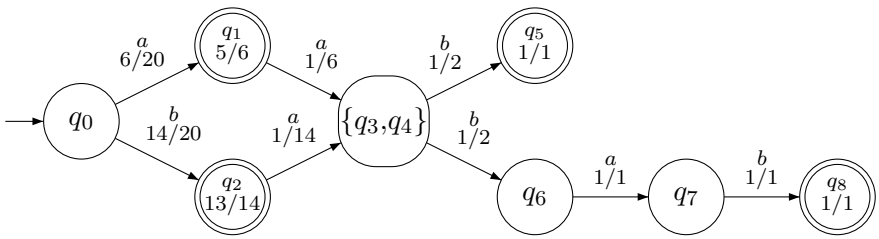


Fig. 6.2 – *Exemple d'automate quotient* $\mathcal{B}$ *: fusions des états* q_3 *et* q_4

exemple, nous pouvons constater que $\mathrm{ct}_\mathcal{B}(\{q_3,q_4\}) = 2$ est bien la somme de $\mathrm{ct}_\mathcal{A}(q_3) = 1$ et $\mathrm{ct}_\mathcal{A}(q_4) = 1$, que $\mathrm{cf}_\mathcal{B}(\{q_3,q_4\}) = 0 = \mathrm{cf}_\mathcal{A}(q_3) + \mathrm{cf}_\mathcal{A}(q_4)$, que $\mathrm{cl}_\mathcal{B}(\{q_3,q_4\},a) = 0 = \mathrm{cl}_\mathcal{A}(q_3,a) + \mathrm{cl}_\mathcal{A}(q_4,a)$ et que $\mathrm{cl}_\mathcal{B}(\{q_3,q_4\},b) = 2 = \mathrm{cl}_\mathcal{A}(q_3,b) + \mathrm{cl}_\mathcal{A}(q_4,b)$

De plus, nous avons le cas typique où l'automate résultant d'une fusion d'états n'est pas déterministe. En effet, la condition $\forall q \in Q_\mathcal{B}, \forall a \in \Sigma, |\{q' \in Q_\mathcal{B} : \delta^\bullet_\mathcal{B}(q,a) = q'\}| \leq 1$, n'est pas remplie car $\{q' \in Q_\mathcal{B} : \delta^\bullet_\mathcal{B}(q,b) = q'\} = \{q_5,q_6\}$. Pour obtenir enfin un automate déterministe, l'opération consiste à fusionner de proche en proche les états introduisant l'indéterminisme. Dans le cas de l'exemple, l'ensemble des états q_5 et q_6 va donc former un nouveau bloc dans une nouvelle partition et l'opération de fusion va être effectuée à nouveau. Le résultat ainsi obtenu (figure 6.3) est quant à lui déterministe : chaque état

[3] Pour que les compteurs $\mathrm{cl}_\mathcal{A}$, $\mathrm{cf}_\mathcal{A}$, $\mathrm{ct}_\mathcal{A}$ soient définis nous considérons que ce DPFA est obtenu depuis un PPTA ayant subi des fusions, ou directement un PPTA.

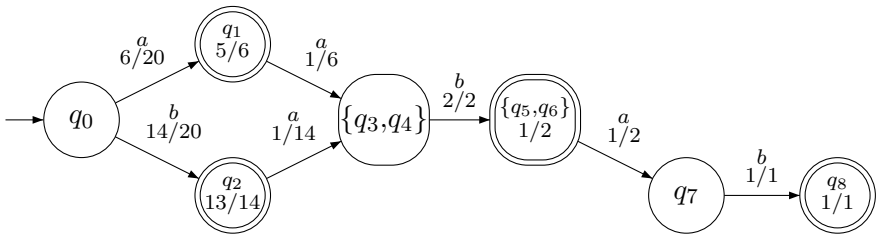


FIG. 6.3 – *Fusion et déterminisation : automate* $\mathcal{C}$

mène par toutes ses transitions sortantes étiquetées avec la même lettre à, au plus, un seul et même état. De la même façon que précédemment, $\mathrm{ct}_{\mathcal{C}}(\{q_5,q_6\}) = 2$ est bien la somme de $\mathrm{ct}_{\mathcal{B}}(q_5) = 1$ et $\mathrm{ct}_{\mathcal{B}}(q_6) = 1$ et les autres compteurs ont été mis à jour de la même manière.

L'étape complète pour passer dans cet exemple du PPTA à l'automate $\mathcal{C}$ est une instance de l'opération atomique « fusion et déterminisation ».

6.4 Espace de recherche

Comme nous l'avons déjà indiqué, le processus générique des algorithmes par fusion d'états prend en entrée l'arbre des préfixes probabilistes et applique sur celui-ci diverses fusions d'états. À chaque étape le langage est ainsi généralisé. Si l'algorithme ne contient pas de critère d'arrêt suffisant, l'étape ultime de fusion nous rend un automate à un seul état : l'automate universel.

Définition 6.4 (Automate universel probabiliste) *Nous notons* UA*, pour* automate universel probabiliste*, un automate dont le nombre d'états vaut exactement* 1*.*

> *Remarque :* Toutes les transitions de cet automate sont des boucles du seul état sur lui même. Un automate universel déterministe probabiliste ne contient donc au plus que $|\Sigma|$ transitions.
>
> Il existe une multitude de ces automates, suivant les probabilités affectées aux transitions. Mais en considérant l'opération de fusion et déterminisation, il existe un seul UA quotient d'un DPFA quelconque.

L'exemple de la figure 6.4 montre l'unique automate universel déterministe probabiliste quotient du PPTA de la figure 6.1 ou d'un de ses quotients comme l'automate $\mathcal{C}$ de la figure 6.3.

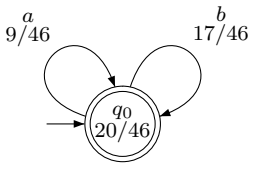


FIG. 6.4 – *Automate universel*

Dans le processus qui nous importe, le PPTA d'un côté et l'UA de l'autre montrent deux apprentissages « extrêmes » non satisfaisants. L'un se contente d'apprendre par cœur les données d'entrée, l'autre accepte toutes les chaînes de Σ^*. L'automate stochastique recherché, construit par fusions d'états, qui nous intéresse se trouve donc quelque part entre le PPTA et l'UA en se déplaçant de l'un à l'autre par fusions successives. L'ensemble où le DPFA cible est recherché est appelé *espace de recherche*.

Propriété 6.1 *L'ensemble des automates quotients d'un automate $\mathcal{A}$ jusqu'à l'automate universel est un treillis [HS66] dont $\mathcal{A}$ est l'élément le plus spécifique et l'*UA *le plus général.*

Nous avons donc besoin, pour apprendre un automate intéressant, de trouver quelles sont les « bonnes » fusions à effectuer depuis le PPTA afin d'obtenir un langage suffisamment généralisé. Nous présentons ci-après l'algorithme générique utilisé dans les processus de généralisation par fusions d'états.

Nous présentons dans la figure 6.5 le treillis complet composé d'un automate représentant en structure le langage composé des mots suivant l'expression régulière : ba^*b, c.-à-d. commençant par un b suivi d'un nombre quelconque de a et finissant par un b, et de tous ses automates quotients. Pour des questions de clarté, les probabilités ne sont volontairement pas présentées.

6.5 Algorithme générique

La plupart des travaux concernant les algorithmes par fusions d'états d'un automate fonctionnent sur le même principe. Les données d'entrée sont les p mots d'un échantillon stochastique S_p, et, en sortie, nous obtenons un DPFA compatible avec les données d'entrées, et de plus généralisé.

Le processus complet est décrit dans l'algorithme 6.1, et se décrit de la façon suivante. Initialement, les données d'entrées sont représentées sous la forme d'un PPTA.

Algorithme 6.1 : Algorithme générique par fusions d'états d'un automate

```
Données : un échantillon stochastique : S_p
Résultat : un DPFA compatible avec S_p
début
    A = PPTA_{S_p};
    tant que (i,j) = Choix_États(A) faire
        si Compatibles(i, j) alors
            A = Fusion_Et_Déterminisation(A, i, j);
        fin
    fin
    retourner A;
fin
```

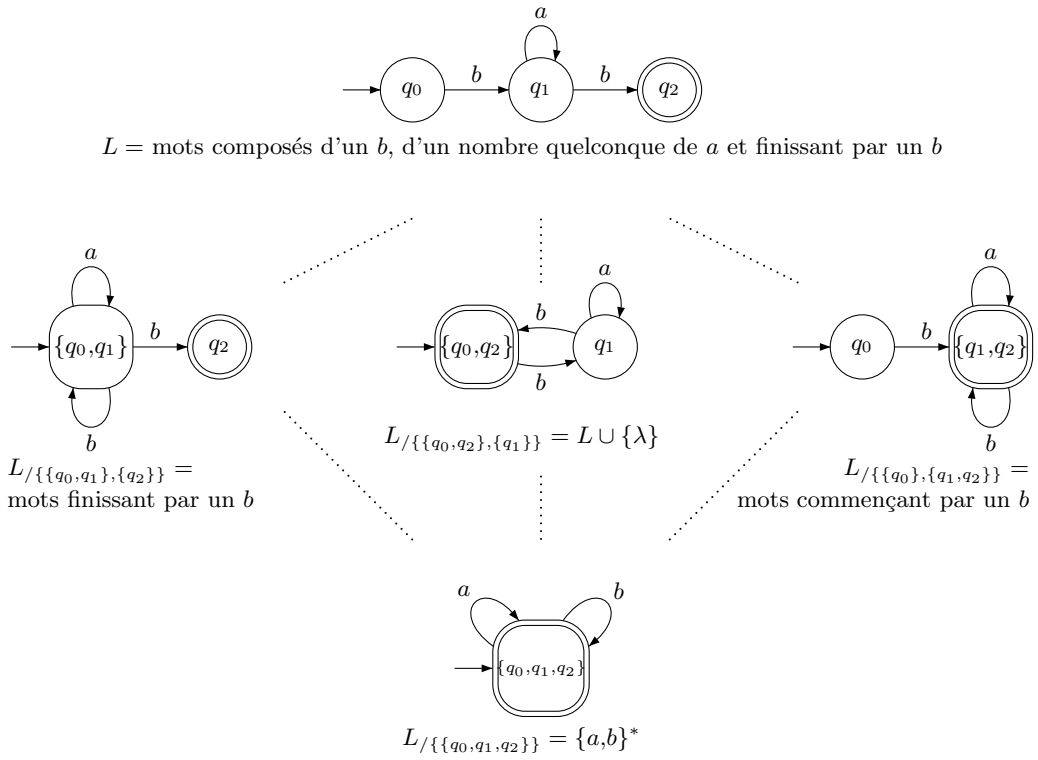


FIG. 6.5 – *Espace de recherche : treillis d'automates*

Ensuite, il faut tester, pour tout couple d'états, s'ils sont « compatibles » et dans ce cas uniquement, les fusionner et déterminiser le résultat obtenu. Quand tous les couples sont testés, l'algorithme renvoie le DPFA courant.

Cet algorithme générique ne sera réellement complet qu'une fois fixés le critère de compatibilité des états à fusionner et l'ordre dans lequel les couples d'états sont choisis. En effet, s'il n'est pas difficile de voir l'importance du critère de compatibilité, il n'en est pas de même pour l'ordre de sélection des couples d'états.

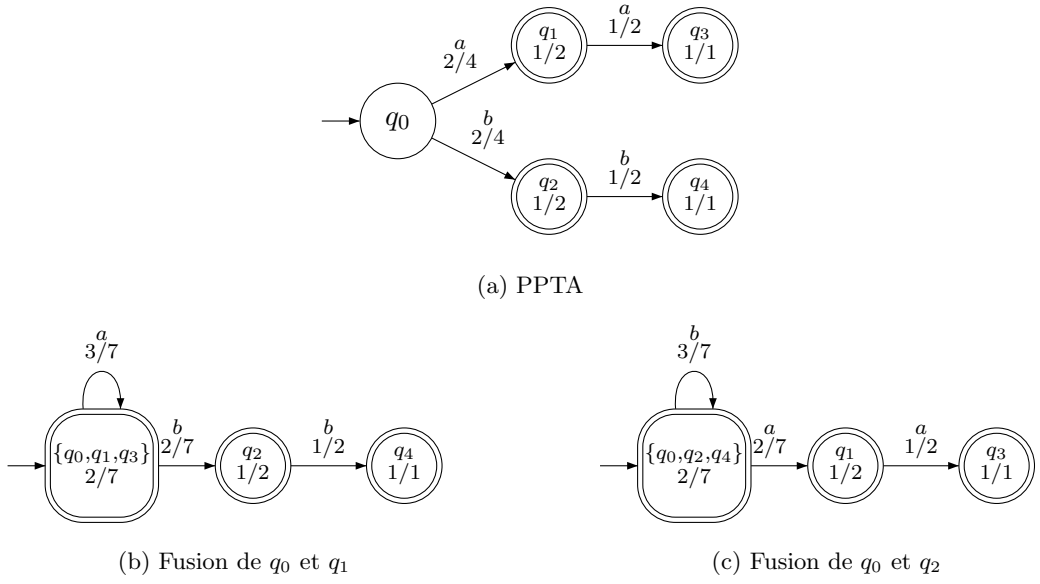
Pour montrer comment le critère de compatibilité peut être prépondérant dans l'algorithme, considérons le PPTA de la figure 6.1 et considérons aussi deux critères de compatibilité pour les états : notons $c_1(q,q')$ (respectivement $c_2(q,q')$) le premier (resp. second) critère définit ainsi :

$$c_1(q,q') = \begin{cases} \text{Vrai} & \text{si } \forall a \in \Sigma, \\ & \mathrm{p}^\bullet(q,a) = \mathrm{p}^\bullet(q',a) \\ \text{Faux} & \text{sinon.} \end{cases} \qquad c_2(q,q') = \begin{cases} \text{Vrai} & \text{si } \forall a \in \Sigma, \\ & \mathrm{p}^\bullet(q,a) = \mathrm{p}^\bullet(q',a) \\ & c_2(\delta^\bullet(q,a), \delta^\bullet(q',a)) \\ \text{Faux} & \text{sinon.} \end{cases}$$

Dans le cas du critère c_1, qui ne prend en considération que les transitions sortantes des états concernés, les états q_3, q_4, q_7, sont compatibles deux à deux. Dans le cas de c_2,

qui requiert de manière supplémentaire que le critère soit récursif, seul le couple (q_3,q_7) reste un choix valide pour la fusion. Ainsi, ce point de l'algorithme est très important et nous expliquerons par la suite les algorithmes connus basés sur des critères différents.

L'ordre de fusion des états est lui aussi important. Il est plus difficile de se faire à l'idée qu'une fusion puisse être possible entre deux états s'ils sont choisis à un instant donné, et pas s'ils sont choisis plus tard. En fait, ce cas n'est possible que si au moins un des deux états a subi une fusion entre-temps. Pour fixer les idées, nous montrons dans la figure 6.6, que le résultat obtenu par fusions successives n'est pas le même suivant l'ordre des fusions considéré. Dans cet exemple, le critère utilisé est c_1. L'automate de



(a) PPTA

(b) Fusion de q_0 et q_1

(c) Fusion de q_0 et q_2

Fig. 6.6 – *De l'importance de l'ordre de sélection des couples d'états*

la figure 6.6(b) est celui obtenu après fusion des états q_0 et q_1, qui sont c_1-compatibles, et déterminisation. Celui de la figure 6.6(c) est obtenu par la fusion de q_0 et q_2, eux aussi c_1-compatibles. Les deux résultats n'ont pas d'états compatibles, l'algorithme est donc terminé. Les résultats obtenus sont structurellement différents : le premier DPFA définit une distribution sur les chaînes de la forme a^*bb et le second sur les chaînes de la forme b^*aa. Dans la suite, nous présentons certains travaux focalisés sur l'importance de l'ordre des fusions.

6.6 Ordre des tests

L'ordre hiérarchique initialement présenté et communément adopté montre [Lan92] des résultats qui peuvent être catastrophiques si la quantité de données initiales n'est pas suffisante. Rappelons que pour apprendre les DPFA il faut au moins un échantillon de taille exponentielle par rapport à la cible. Ainsi, de nombreux travaux montrent qu'il

est possible d'obtenir de meilleurs résultats en choisissant de manière appropriée les états à fusionner. Tout d'abord, l'adaptation *evidence driven* [LPP98] de l'algorithme générique par fusions d'états est basée sur la fusion du couple d'états maximisant un critère défini. Cette version a une complexité élevée : en effet il faut tester tous les couples possibles avant la fusion. Une version algorithmiquement moins lourde est l'adaptation *data driven* [HOV96], qui fusionne le couple d'états compatibles par lequel passe le plus d'information (c.-à-d. les états ayant les compteurs ct les plus élevés). L'algorithme *Blue-Fringe* [LPP98] restreint l'ensemble des couples à tester mais garde le principe de l'adaptation *evidence driven* : Lang définit des classes d'états rouges, bleus et blanc correspondant respectivement à des états « consolidés » (ne pouvant plus être fusionnés avec des états de la même classe), « candidats » et non traités. Les fusions ne sont évaluées que dans le cas où le couple choisi contient un état rouge et un bleu.

6.7 Critères de compatibilité et algorithmes

Le critère de compatibilité des états est le cœur même de l'algorithme. En effet, la décision de fusionner deux états sera prise en fonction de l'évaluation de ce test ; elle est définitive : aucun retour en arrière n'est prévu dans l'algorithme générique. Il est donc important de ne fusionner que des états extrêmement proches en terme de langages stochastiques engendrés (c'est à dire langage obtenu si nous considérons l'état considéré comme l'état initial).

Les divers travaux présentant des algorithmes d'inférence se focalisent sur le critère de compatibilité. Ainsi, dans la suite, nous présentons deux algorithmes souvent utilisés avec le critère qui leur est associé.

Nous détaillons dans cette partie deux algorithmes par fusions d'états introduits dans le cadre de l'inférence grammaticale régulière stochastique : tout d'abord, Alergia développé par Carrasco et Oncina, puis un algorithme se comportant mieux en reconnaissance de la parole, Mdi présenté par Thollard, Dupont et De La Higuera.

6.7.1 Alergia

L'algorithme Alergia [CO94a], repose sur un principe simple : après avoir choisi un couple d'états, ils sont fusionnés si et seulement si les langages qu'ils engendrent sont assez proches. Le critère de fusion sera donc un test statistique récursif sur les états choisis et leurs successeurs (états atteignables par une suite de transitions depuis les états choisis).

L'idée du test est d'appréhender chaque transition sortante, étiquetée par une lettre quelconque a, d'un état q, munie d'une probabilité $\mathrm{p}^\bullet(q,a) = \frac{\mathrm{cl}(q,a)}{\mathrm{ct}(q)}$, comme une variable statistique de Bernoulli. Ce type de variable modélise les succès/erreurs lors d'une expérience : le cas le plus typique est un lancer de pièce de monnaie, le côté « pile » est considéré comme un succès, l'autre côté « face » est un échec.

Considérons une variable aléatoire, nous appelons essai de Bernoulli une expé-

rience conduisant à un succès avec probabilité[4] p ou à un échec avec probabilité $1-p$. Dans [Hoe63], en considérant n essais, ayant conduits à k succès, une borne d'erreur d'estimation de probabilité théorique p est donnée en fonction d'un paramètre α : avec une probabilité $1-\alpha$, la différence entre p et le rapport[5] du nombre de succès k sur le nombre d'essais n est inférieure à une constante. Nous le réécrivons de manière formelle :

$$\left| p - \frac{k}{n} \right| < \sqrt{\frac{1}{2} \times \ln\left(\frac{2}{\alpha}\right)} \times \frac{1}{\sqrt{n}}$$

Dans le cas de l'algorithme Alergia, les auteurs fusionnent deux états si les estimations sur chaque lettre et sur le fait d'être final sont toutes inférieures à la somme des erreurs d'estimation suivant la borne de Hoeffding. Pour cela, la variable modélisant une transition sortant d'un état q sur la lettre a a comme nombre de succès $\mathrm{cl}(q,a)$ et celle modélisant le fait d'être final $\mathrm{cf}(q)$; le nombre d'essais est $\mathrm{ct}(q)$. La fonction « Compatibles » de l'algorithme générique est réécrite pour Alergia dans la fonction 6.2. Dans le cas où les états q et q' passent le test de compatibilité présenté,

Fonction `CompatiblesAlergia`

Données : un automate : $\mathcal{A}$, deux états : `q` et `q'`, un paramètre : α
pour chaque $a \in \Sigma$ **faire**
  si $\left|\frac{\mathrm{cl}(\mathsf{q},a)}{\mathrm{ct}(\mathsf{q})} - \frac{\mathrm{cl}(\mathsf{q}',a)}{\mathrm{ct}(\mathsf{q}')}\right| > \sqrt{\frac{1}{2} \times \ln\left(\frac{2}{\alpha}\right)} \times \left(\frac{1}{\sqrt{\mathrm{ct}(\mathsf{q})}} + \frac{1}{\sqrt{\mathrm{ct}(\mathsf{q}')}}\right)$ **alors retourner** Faux;
fin
si $\left|\frac{\mathrm{cf}(\mathsf{q})}{\mathrm{ct}(\mathsf{q})} - \frac{\mathrm{cf}(\mathsf{q}')}{\mathrm{ct}(\mathsf{q}')}\right| > \sqrt{\frac{1}{2} \times \ln\left(\frac{2}{\alpha}\right)} \times \left(\frac{1}{\sqrt{\mathrm{ct}(\mathsf{q})}} + \frac{1}{\sqrt{\mathrm{ct}(\mathsf{q}')}}\right)$ **alors retourner** Faux;
pour chaque $a \in \Sigma$ **faire**
  si `non(CompatiblesAlergia(`$\mathcal{A}$, $\delta^\bullet(\mathsf{q},a),\delta^\bullet(\mathsf{q}',a),\alpha$`))` **alors retourner** Faux;
fin
retourner Vrai;

nous disons qu'ils sont α-compatibles.

De manière théorique, les auteurs de Alergia montrent [CO99] qu'une version allégée de l'algorithme, baptisée Rlips, identifie à la limite la cible avec probabilité 1. Ensuite, De La Higuera et Thollard étendent [HT00] la preuve à l'algorithme initial.

Un des problèmes de cet algorithme est qu'il ne fusionne pas forcément les états les plus proches avec le critère de fusion par défaut et l'ordre hiérarchique de sélection des états. Divers travaux [KD02, HBS03], focalisés sur le critère de fusion, montrent de meilleurs résultats pratiques : d'abord avec un test multinomial, puis une adaptation plus restrictive du critère de fusion.

[4] dans le cas de la pièce de monnaie non pipée, $p = 0{,}5$
[5] cette grandeur est l'estimation empirique de p

6.7.2 MDI

Le test de compatibilité de l'algorithme ALERGIA, bien que récursif, est local. En effet, seuls sont comparés les langages engendrés par les deux états choisis pour la fusion, alors que la distribution complète sur toutes les chaînes peut être modifiée. L'algorithme MDI prend en considération les différences sur le langage stochastique complet.

Le critère de compatibilité de cet algorithme est basé sur l'approche bayésienne qui consiste à trouver le modèle qui maximise la vraisemblance des données d'apprentissage S_p. D'après la formule de BAYES, cela revient à maximiser la probabilité *a posteriori* du modèle sachant les données d'apprentissage (ce qui revient à les apprendre par cœur) et la probabilité *a priori* du modèle défini dans ce cadre comme inversement proportionnelle à sa taille en nombre d'états ; la probabilité *a priori* de S_p étant constante, elle n'intervient pas dans la maximisation. Nous invitons le lecteur intéressé par le choix de l'inverse de la taille comme probabilité *a priori* à lire [Tho00] qui introduit le principe du rasoir d'OCCAM.

À chaque fusion, nous avons vu que le langage se généralise, il s'éloigne donc de la distribution initiale. Le critère de compatibilité devient donc un compromis entre éloignement des distributions et compacité du modèle appris. En considérant un automate $\mathcal{A}$, deux états et un paramètre α, nous notons $\mathcal{B}$ le DPFA quotient obtenu après fusion de ces états et déterminisation. Les états sont α-compatibles si le rapport entre l'éloignement des distributions ($\mathrm{d}_{KL}(\mathcal{A}, \mathcal{B})$) et la différence de tailles des automates ($|Q_{\mathcal{A}} - Q_{\mathcal{B}}|$) est inférieur à α.

La fonction « Compatibles » de l'algorithme générique est réécrite pour MDI dans la fonction 6.3.

Fonction `CompatiblesMdi`

Données : un automate : $\mathcal{A}$, deux états : `q` et `q'`, un paramètre : α
$\mathcal{B}$ = `Fusion_Et_Déterminisation(`$\mathcal{A}$`, q, q')`;
$\mathsf{div} = \mathrm{d}_{KL}(\mathcal{A}, \mathcal{B})$;
$\Delta\mathsf{S} = |Q_{\mathcal{A}} - Q_{\mathcal{B}}|$;
retourner $\frac{\mathsf{div}}{\Delta\mathsf{S}} < \alpha$;

L'algorithme MDI a été conçu dans le cadre de la thèse de F. THOLLARD [Tho00] ayant pour sujet le traitement automatique de la langue naturelle. Dans ce cadre là, les résultats pratiques obtenus [TDH00, TD00] sur des bases de tests connus (ATIS [Hir92]) sont meilleurs avec le test global de MDI que celui d'ALERGIA.

6.8 Conclusion ⋆

Dans ce chapitre, nous avons présenté un algorithme générique d'inférence grammaticale stochastique par fusions d'états. Celui-ci définit la trame de toute une famille d'algorithmes dits à fusions d'états. Il se déroule en plusieurs étapes : l'étape initiale est

rendue possible grâce à la construction du PPTA ; les emplois successifs de l'opération de fusion et déterminisation permettent d'une part de généraliser le modèle et d'autre part d'obtenir un modèle déterministe. Le choix de l'ordre de sélection des états pour cette opération est crucial, nous expliquons comment améliorer les performances en changeant cet ordre. De plus, nous donnons deux critères de compatibilité entre états pour la fusion, pour décrire deux algorithmes issus de l'algorithme générique.

7 Critères d'évaluation

Sommaire

Résumé

Les algorithmes d'apprentissage automatique génèrent une hypothèse en fonction des données de départ. Un des problèmes importants et difficiles en inférence grammaticale est d'évaluer le modèle appris en terme de performances. Nous devons donc définir la qualité d'un modèle appris. Deux possibilités se présentent : si la distribution de probabilité cible est connue (cas rares) il suffit de « mesurer » l'écart entre cette dernière et la distribution représentée par le modèle appris ; dans le cas où la cible est inconnue, nous estimerons cet écart sur un échantillon d'exemples de tests.

7.1 Introduction ⋆

L'un des problèmes difficiles de l'inférence grammaticale stochastique est de pouvoir évaluer la qualité de l'apprentissage. En effet, dans la plupart des cas, la cible $\mathfrak{D}$ n'est pas connue et lorsqu'aucune donnée négative n'est disponible, tous les modèles appris acceptant les exemples avec une probabilité similaire à celle de l'échantillon d'apprentissage sont des candidats potentiels. Pourtant ils ne se valent pas. Le protocole largement adopté dans ce cas est de séparer les données disponibles en deux ensembles : un d'apprentissage et un de test. Ainsi, une fois l'apprentissage effectué, il est possible de tester le modèle appris en estimant la « distance » entre la distribution du test et celle du modèle. Il existe néanmoins un compromis entre la taille des différents ensembles : statistiquement, les données étant tirées en suivant une distribution $\mathfrak{D}$, plus l'ensemble de test sera grand et plus il représentera fidèlement la distribution $\mathfrak{D}$. Il faudrait donc garder une grande partie des données dans l'ensemble de test. À l'opposé, l'apprentissage stochastique a besoin d'une grande quantité de données pour apprendre la structure de la cible mais surtout pour estimer au mieux les probabilités associées

à chaque transition d'un DPFA. Ainsi, il est souvent intéressant d'avoir une grande quantité de données disponibles pour mener à bien, l'apprentissage et l'évaluation des modèles appris par le biais de ce dernier.

Cette évaluation est souvent établie dans la littérature grâce à la perplexité, une grandeur qui estime en partie la divergence de Kullback-Leibler quand la cible n'est pas connue mais seulement estimée. Nous définissons formellement cette grandeur en expliquant succinctement ce qu'elle représente de différents points de vues.

7.2 Perplexité

En considérant la divergence de Kullback-Leibler entre deux distributions $\mathfrak{D}_{\mathrm{C}}$ et $\mathfrak{D}_{\mathrm{M}}$, ou plus précisément l'entropie de $\mathfrak{D}_{\mathrm{C}}$ relativement à $\mathfrak{D}_{\mathrm{M}}$, nous possédons une mesure permettant l'évaluation d'un modèle appris représentant $\mathfrak{D}_{\mathrm{M}}$ pour une distribution cible $\mathfrak{D}_{\mathrm{C}}$. Malheureusement, dans la plupart des cas, le modèle de langage cible n'est pas connu. Ainsi, il est intéressant d'estimer cette grandeur en prenant en compte les éléments du langage présents dans l'ensemble de test plutôt que la distribution cible inconnue. Comme le but est de trouver le meilleur modèle de langage appris $\mathfrak{D}_{\mathrm{M}}$ pour une distribution cible donnée , nous considérons que $\mathfrak{D}_{\mathrm{C}}$ ne change pas avec le temps.

La divergence s'écrit ainsi :

$$\mathrm{d}_{KL}(\mathfrak{D}_{\mathrm{C}},\mathfrak{D}_{\mathrm{M}}) = \sum_{w\in\Sigma^*} \mathrm{P}_{\mathfrak{D}_{\mathrm{C}}}(w) \log \frac{\mathrm{P}_{\mathfrak{D}_{\mathrm{C}}}(w)}{\mathrm{P}_{\mathfrak{D}_{\mathrm{M}}}(w)}$$

ou encore :

$$\mathrm{d}_{KL}(\mathfrak{D}_{\mathrm{C}},\mathfrak{D}_{\mathrm{M}}) = \sum_{w\in\Sigma^*} \big(\mathrm{P}_{\mathfrak{D}_{\mathrm{C}}}(w) \log \mathrm{P}_{\mathfrak{D}_{\mathrm{C}}}(w) - \mathrm{P}_{\mathfrak{D}_{\mathrm{C}}}(w) \log \mathrm{P}_{\mathfrak{D}_{\mathrm{M}}}(w)\big)$$

La première partie de la somme étant constante, nous ne nous intéressons qu'à la deuxième partie. Considérons un multi-ensemble de test $\mathrm{T} = \{w_1,w_2,\ldots,w_{|T|}\}$. En estimant $\mathfrak{D}_{\mathrm{C}}$ par la distribution empirique de l'ensemble de test[1], notée $\mathfrak{D}_{\mathrm{T}}$; la partie de la formule qui nous intéresse devient :

$$-\frac{1}{|T|}\sum_{i=1}^{|T|} \log \mathrm{P}_{\mathfrak{D}_{\mathrm{M}}}(w_i)$$

En rappellant la définition itérative du calcul d'une probabilité d'un mot dans un automate :

$$\mathrm{p}(q,w) = \prod_{i=1}^{|w|} \Big(\mathrm{p}^{\bullet}\big(\delta\big(q, \underset{i-1}{\mathrm{Pr}}(w)\big),w[i]\big)\Big) \times \mathrm{f}\big(\delta(q,w)\big)$$

[1] $\widehat{\mathrm{P}_{\mathfrak{D}_{\mathrm{C}}}(w)} = \mathrm{P}_{\mathfrak{D}_{\mathrm{T}}}(w) = \frac{\mathrm{Occ}(w)}{|T|}$

nous obtenons (car le log d'une somme d'éléments est égal à la somme des logs des mêmes éléments) :

$$-\frac{1}{|T|}\sum_{i=1}^{|T|}\left(\sum_{j=1}^{|w_i|}\log\left(\mathrm{p}^{\bullet}\left(\delta\big(q_0,\Pr_{j-1}(w_i)\big),w_i[j]\right)\right)+\log\mathrm{f}\big(\delta(q_0,w_i)\big)\right)$$

En remarquant que la somme ne concerne que des probabilités de transitions ou d'arrêt, et en notant $\mathrm{p}(w,i)=\mathrm{p}^{\bullet}\left(\delta\big(q_0,\Pr_{i-1}(w)\big),w[i]\right)$ pour $1\leq i\leq|w|$ et $\mathrm{p}(w,|w|+1)=\mathrm{f}\big(\delta(q_0,w)\big)$, on obtient :

$$-\frac{1}{|T|}\sum_{i=1}^{|T|}\sum_{j=1}^{|w_i+1|}\log\mathrm{p}(w_i,j)$$

De manière pratique, la définition de la perplexité est basée sur les symboles que constituent les mots de T. Nous ordonnons ces symboles en numerotant la première lettre du premier mot comme 1, la deuxième lettre du premier mot comme 2, et ainsi de suite. La dernière lettre du premier mot est le symbole numéro |premier mot|. Le symbole suivant est défini comme le fait de finir la lecture d'un mot : ainsi la première lettre du deuxième mot aura pour numéro d'ordre |premier mot| + 2. Nous notons ces symboles s_i où l'indice i est compris entre 1 et la somme des tailles des mots de T en comptant le fait de finir la lecture des chaines, de manière habituel ce nombre est noté $||\,\mathrm{T}\,||$. De la sorte il existe une bijection de $\mathbb{N}^2$ dans $\mathbb{N}$, qui associe la j^{e} lettre du i^{e} mot de T au k^{e} symbole défini précédemment. Nous pouvons noter $\mathrm{p}(s_k)=\mathrm{p}(w_i,j)$.

Ainsi nous obtenons pour la partie de l'estimation de la divergence qui nous intéresse

$$-\frac{1}{|T|}\sum_{k=1}^{||\,\mathrm{T}\,||}\log\mathrm{p}(s_k)$$

De manière classique, l'entropie est souvent vue comme la « taille » d'un modèle en terme de codage informatique, et donc, pour des raisons d'homogénéité, la perplexité est souvent renormalisée en fonction de la taille de l'échantillon de test en terme de nombre de symboles, plutôt qu'en terme de nombre de mots. La définition suivante donne la manière classique de présenter la perplexité.

Définition 7.1 (Perplexité) *Soit une distribution de probabilité $\mathfrak{D}_{\mathrm{C}}$ cible inconnue, un échantillon tiré selon $\mathfrak{D}_{\mathrm{C}}$ appelé* T *et un modèle de langage représenté par un* DPFA, *$\mathcal{A}$; nous définissons la* perplexité *du langage représenté par $\mathcal{A}$ relativement à l'échantillon* T *comme suit :*

$$\mathrm{pp}(\mathcal{A}\,,\mathrm{T})=2^{-\frac{1}{||T||}\sum_{k=1}^{||\,\mathrm{T}\,||}\log\mathrm{p}_{\mathcal{A}}(s_k)}$$

Remarque : Ainsi plus la perplexité est faible, meilleure est l'adéquation entre le modèle de langage et l'échantillon de test. Le problème de cette grandeur, héritée de la divergence de Kullback-Leibler, est qu'elle n'est pas définie, du

moment où il existe un élément de l'ensemble de test qui ne peut être reconnu par l'automate : dans ce cas, la probabilité de transition (ou d'être final) est nulle et la perplexité est alors infinie.

Cette grandeur est très utilisée dans la communauté de reconnaissance de formes et en particulier en reconnaissance vocale. De plus, la quasi-totalité des résultats d'apprentissage stochastique y font référence.

Pour pallier au problème de la définition des grandeurs d_{KL} et pp dans les cas où elles deviennent infinies, nous avons recours au lissage, procédé dont le principe est de modifier légèrement le modèle d'apprentissage pour qu'il donne une probabilité non nulle à chacun des mots possibles.

7.3 Lissage

Nous avons vu que la perplexité devient infinie dans le cas où le modèle de langage n'accepte pas un mot de l'échantillon de test. Les modèles appris qui sont fidèles aux données d'apprentissage peuvent ne pas contenir des mots de l'échantillon de tests. Plus l'ensemble d'apprentissage est grand et/ou plus la généralisation au cours de l'apprentissage a été grande alors moins ce phénomène est probable. En effet, pour qu'un mot ne soit pas accepté pour le modèle appris, il peut y avoir deux cas :

- tout d'abord, il se peut qu'apparaissent dans l'échantillon de test des lettres non présentes dans l'ensemble d'apprentissage. Par exemple, imaginons un modèle de langage sur l'aphabet a,b,c avec une distribution de probabilité faible mais non nulle sur les mots contenant la lettre c. L'ensemble d'apprentissage, quelle que soit sa taille, peut ne pas représenter de mot contenant de c.
- puis, même si nous considérons l'alphabet union de ceux représentés par les ensembles d'apprentissage et de test, il se peut qu'un mot de l'ensemble de test ne soit pas accepté par le modèle car il n'appartient pas à l'ensemble de départ et que la généralisation a été trop faible.

Dans tous les cas, en supposant les ensembles respectifs de taille suffisamment importante, les mots qui posent problèmes sont des mots qui ont une probabilité faible dans la distribution recherchée. Le principe de lissage est donc d'affecter à ces mots une probabilité infime mais non nulle, en gardant les propriétés de consistance pour une distribution de probabilité, à savoir que la somme des probabilités des mots de Σ^* soit égale à 1.

Pour cela, nous utilisons deux méthodes distinctes que nous présentons : l'interpolation linéaire du modèle appris avec un autre plus général, et une méthode par repli qui fixe la probabilité d'un mot en fonction de son acceptation ou non par le modèle appris.

7.3.1 Interpolation linéaire de modèles de langages

Le principe est assez simple : certains mots de Σ^* ont une probabilité nulle dans le modèle de langage appris, soit parce qu'un caractère les composant n'appartient pas à l'ensemble de test, soit parce que la généralisation de ce dernier ensemble n'accepte pas le mot. Dans ce cas, la probabilité que le mot fasse partie de la distribution cible est faible mais pas forcément nulle. L'idée est de se servir d'un modèle le plus général possible (c.-à-d. représenté par un automate universel[2]) pour le combiner linéairement avec le modèle appris afin d'obtenir que tous les mots aient une probabilité strictement positive. La loi de probabilité se définit ainsi :

$$\mathrm{P}(w) = \beta.\,\mathrm{P}_{\mathfrak{D}_\mathrm{M}}(w) + (1-\beta).\,\mathrm{P}_{\mathfrak{D}_\mathrm{UA}}(w)$$

où $\beta \in [0,1[$ représente le paramètre du lissage, $\mathrm{P}_{\mathfrak{D}_\mathrm{M}}$ représente la loi de probabilité associée au modèle appris, et $\mathfrak{D}_\mathrm{UA}$ la loi de probabilité de l'automate universel. Les probabilités de l'automate universel utilisées sont estimées sur l'échantillon d'apprentissage en ajoutant une transition qui accepte n'importe quel symbole non vu, et en renormalisant ensuite pour garder une distribution de probabilités.

Avec cette méthode de lissage, quel que soit le mot considéré, la probabilité associée n'est jamais nulle. De plus, le paramètre β nous permet de fixer la confiance que l'on peut avoir dans l'apprentissage : en effet, plus β est proche de 1, plus le poids du modèle est important dans la combinaison linéaire obtenue.

Cette méthode simple est largement utilisée dans la littérature. De fait, le lissage change donc la distribution de probabilité apprise, si de plus le paramètre β est faible, la probabilité résultant de la combinaison peut même être considérablement éloignée de celle apprise. Ainsi, en plus de l'apprentissage, le lissage permet de modifier la distribution. Il paraît alors évident d'étudier le lissage pour améliorer les modèles. Un sous-domaine entier de l'apprentissage automatique s'est développé pour comparer théoriquement et empiriquement les améliorations apportées par un bon lissage. Certaines méthodes statistiques sophistiquées laissent la généralisation complète au lissage et présentent donc comme modèle appris, le PPTA. Sans aller jusqu'à ce cas extrême, qui ne distingue plus l'apprentissage automatique du lissage, il existe des méthodes donnant de meilleurs résultats. Nous présentons ci-après une méthode dite « par repli ».

7.3.2 Méthode par repli

Quelle que soit la probabilité assignée par le modèle appris, l'interpolation linéaire renvoie une probabilité issue de deux modèles $\mathfrak{D}_\mathrm{M}$ et $\mathfrak{D}_\mathrm{UA}$. La méthode par repli, telle que nous l'utilisons, initialement présentée par Katz [Kat87] est différente en ce sens : nous considèrons que les chaînes non reconnues par le modèle appris représentent une certaine quantité dans la masse de probabilité qu'il faut redistribuer sur les chaînes

[2] Nous rappelons au lecteur, qu'un automate universel accepte tous les mots de Σ^*

concernées pour garder une distribution avec une somme de probabilités égale à 1. En reprenant les mêmes notations, dans ce cas, la probabilité d'un mot est définie ainsi :

$$\mathrm{P}(w) = \begin{cases} \mathrm{P}_{\mathfrak{D}_\mathrm{M}}(w) - \beta & \text{si} \quad \mathrm{P}_{\mathfrak{D}_\mathrm{M}}(w) \neq 0 \\ c_\beta . \mathrm{P}_{\mathfrak{D}_\mathrm{UA}}(w) & \text{sinon} \end{cases}$$

où β représente le paramètre du lissage (qui est une certaine quantité de masse de probabilité à redistribuer), et c_β représente un coefficient de renormalisation de telle sorte que la somme des probabilités de tous les mots soit bien 1. Lorsque nous parlons dans les expérimentations de méthode par repli, c'est de cette formule qu'il s'agit. Divers travaux ont montré empiriquement que les méthodes par repli peuvent être améliorées et surclassent les méthodes par interpolation.

Pour conserver une grande importance au modèle appris, nous nous contenterons de méthodes de lissage simple : interpolation avec un automate universel, ou méthode de repli simple avec le même automate universel. Le choix de la méthode utilisée est, pour nous, soit fixé lorsque nous nous comparons à des résultats existant, soit fait de manière à ne pas trop changer le modèle appris pour se comparer facilement à une méthode à base de distance ne nécessitant pas l'utilisation de modèle lissé.

Les critères d'évaluation de l'apprentissage automatique que nous présentons sont basés soit sur l'utilisation de la perplexité, soit d'une distance d'automates dont nous parlons dans le chapitre suivant : les expérimentations présentées dans ce chapitre montrent en quoi l'utilisation des distances est plus intéressante pour nous que celle de la perplexité.

7.4 Conclusion ⋆

La perplexité est donc la mesure inconstestée du domaine dans la littérature. Nous avons dans ce chapitre défini de manière formelle cette grandeur, puis montré comment lisser un modèle — condition *sine qua non* pour obtenir une perplexité finie. Les deux méthodes de lissage présentées sont très simples, mais nous invitons le lecteur intéressé à lire [CG99][3] pour une étude comparative de diverses méthodes de lissage sur les modèles de langages. Dans le chapitre suivant, nous présentons une nouvelle mesure entre automates qui permettra d'évaluer les modèles appris sans les lisser.

3 ou mieux un rapport antérieur mais plus complet [CG98]

8 Distances entre modèles de langages ⋆

Sommaire

Résumé

Nous voulons définir une vraie distance, au sens mathématique, entre les modèles de langages, qui ne souffre pas de problème de la divergence de Kullback-Leibler. Dans ce chapitre, nous définissons donc formellement une mesure entre distributions de probabilité, nous montrons qu'elle correspond à une distance et qu'elle est calculable sur les DPFA. De plus, nous menons quelques expérimentations pour montrer le comportement de cette mesure dans le cadre de l'apprentissage automatique et celui général de la classification.

8.1 Introduction ⋆

Nous avons introduit dans le chapitre 4 la notion de distance entre distributions de probabilité. Nous allons nous intéresser maintenant de manière formelle à la distance euclidienne sur des représentants que sont les DPFA.

Comme nous l'avons déjà vu, la mesure de similarité souvent utilisée dans ces domaines vient de la reconnaissance de formes et mesure une différence d'entropie. La divergence de Kullback-Leibler souffre de plusieurs points négatifs : le plus important en inférence grammaticale est la nécessité de lisser les modèles appris pour obtenir une d_{KL} toujours définie. Un autre point négatif est que cette mesure n'est pas une distance au sens mathématique du terme, ceci empêche toute opération topologique sur l'espace des distributions.

Dans ce chapitre nous redonnons la définition de la distance d_2 sur les DPFA, puis nous montrons comment la calculer efficacement. Nous montrons enfin son comportement expérimental sur de multiples tâches : identification, robustesse à l'échantillonnage (avec une vitesse de convergence rapide), classification et évaluation de modèles appris.

8.2 Définition ⋆

Les premières idées de distances viennent de l'extension de distances connues dans des espaces de dimensions finies — comme $\mathbb{R}^n$ — aux DPFA qui peuvent être vus comme des vecteurs de probabilité sur Σ^* muni de l'ordre lexicographique. Ainsi les distances classiques $\mathrm{d}_1, \mathrm{d}_2, \ldots, \mathrm{d}_n$ paraissent difficilement calculables.

Néanmoins, nous montrons ci-après qu'il existe un développement formel de la d_2 dans ce cas là. Nous proposons une extension de la distance euclidienne « classique » d'un espace de dimension finie aux distributions de probabilités représentées par des DPFA. Nous donnons de nouveau la définition formelle de cette distance, cette fois-ci appliquée aux automates.

Définition 8.1 (Distance euclidienne de DPFA**)** *Soient $\mathcal{A}, \mathcal{B}$ deux automates stochastiques déterministes à états finis définis sur un unique alphabet Σ, nous définissons la* distance euclidienne *entre ces deux entités comme la racine carrée de la somme des carrées des différences des probabilités de chacun des mots de Σ^*.*

$$\mathrm{d}_2(\mathcal{A}, \mathcal{B}) = \sqrt{\sum_{w \in \Sigma^*} \left(p_{\mathcal{A}}(w) - p_{\mathcal{B}}(w)\right)^2}$$

Nous montrons ensuite que cette valeur est toujours définie, que c'est une vraie distance et qu'elle est calculable.

8.3 Résultats théoriques ⋆

Pour expliciter le calcul complet de cette distance, nous aurons besoin du concept de fonction de probabilité préfixielle : la probabilité pour qu'un mot soit un préfixe des mots du langage.

Définition 8.2 (Fonction de probabilité préfixielle) *Soit Σ un alphabet fini, w un mot de Σ, $\mathcal{A}$ un automate, nous notons la* fonction de probabilité préfixielle *de w pour $\mathcal{A}$ par :*

$$\pi_{\mathcal{A}}(w) = \sum_{x \in \Sigma^*} \mathrm{p}_{\mathcal{A}}(wx)$$

Remarque : Cette fonction de probabilité ne définit pas une distribution de probabilité au sens strict du terme, en effet $\sum_{w \in \Sigma^*} \sum_{x \in \Sigma^*} p_{\mathcal{A}}(wx)$ peut être supérieure à 1. Cependant, elle nous permet de présenter facilement certaines manipulations sur le calcul des probabilités dans les automates.

Nous notons ci-après les différents liens utiles entre les fonctions de probabilité et de probabilité préfixielle.

$$\forall w \in \Sigma^*, \quad \pi(w) = \mathrm{p}(w) + \sum_{a \in \Sigma} \pi(wa) \tag{8.1}$$

$$\forall w \in \Sigma^*, \forall a \in \Sigma, \quad \pi(wa) = \pi(w) \times \mathrm{p}^\bullet(\delta(q_0,w),a) \tag{8.2}$$

$$\forall w \in \Sigma^*, \quad \mathrm{p}(w) = \pi(w) \times \mathrm{f}(\delta(q_0,w)) \tag{8.3}$$

Preuve : Les preuves des ces propriétés sont simples et données en annexe 1. □

Le développement complet de la formule de calcul de la distance d_2 entre deux DPFA, $\mathcal{A}$ et $\mathcal{B}$, fait apparaître la probabilité d'atteindre en même temps les états $q \in Q_\mathcal{A}$ et $q' \in Q_\mathcal{B}$ en lisant le même mot $w \in \Sigma^*$. Pour cela nous définissons une grandeur $\eta_{qq'}$ qui la représente. Nous notons $q_{0_\mathcal{A}}$ et $q_{0_\mathcal{B}}$ respectivement les états initiaux de $\mathcal{A}$ et $\mathcal{B}$.

Définition 8.3 *Soient $q \in Q_\mathcal{A}$, $q' \in Q_\mathcal{B}$, nous définissons :*

$$\eta_{qq'} = \sum_{\substack{w \in \Sigma^* : \\ \delta_\mathcal{A}(q_{0_\mathcal{A}},w)=q \\ \delta_\mathcal{B}(q_{0_\mathcal{B}},w)=q'}} \pi_\mathcal{A}(w).\,\pi_\mathcal{B}(w)$$

Lorsque Σ^* n'est pas un ensemble fini — ce qui est souvent le cas, la relation précédente peut paraître difficilement calculable. Nous nous proposons de calculer itérativement les valeurs de $\eta_{qq'}$ pour tous les couples d'états (q,q'). Pour cela, nous nous servirons de la notion de co-émission, introduite par Lyngso *et al.* dans le cas d'une sous-classe de modèles de Markov [LPN99], et nous adapterons la méthode de calcul, présentée dans [CRJ02], de distances sur les arbres.

En sortant λ de l'ensemble Σ^* et en appliquant la propriété donné par l'équation 8.2, nous obtenons la formulation suivante :

$$\eta_{qq'} = \begin{cases} 1 + \eta^*_{qq'} & \text{si } q = q_{0_\mathcal{A}} \text{ et } q' = q_{0_\mathcal{B}} \\ \\ \eta^*_{qq'} & \text{sinon} \end{cases}$$

avec $\eta^*_{qq'} = \sum_{\substack{w \in \Sigma^* : \\ w \neq \lambda \\ \delta_\mathcal{A}(q_{0_\mathcal{A}},w)=q \\ \delta_\mathcal{B}(q_{0_\mathcal{B}},w)=q'}} \pi_\mathcal{A}(w).\,\pi_\mathcal{B}(w).$

De plus, pour chaque mot w de taille strictement positive, il existe un préfixe $x \in \Sigma^*$ et une lettre $a \in \Sigma$ qui le termine, nous réécrivons $w = xa$ dans la formule des $\eta^*_{qq'}$.

$$\eta^*_{qq'} = \sum_{x \in \Sigma^*} \sum_{\substack{a \in \Sigma : \\ \delta_\mathcal{A}(q_{0_\mathcal{A}},xa)=q \\ \delta_\mathcal{B}(q_{0_\mathcal{B}},xa)=q'}} \pi_\mathcal{A}(xa).\,\pi_\mathcal{B}(xa)$$

En considérant tous les couples possibles d'états d'arrivée de la lecture du mot x et non plus xa, nous pouvons réécrire :

$$\eta^*_{qq'} = \sum_{s\in Q_\mathcal{A}} \sum_{s'\in Q_\mathcal{B}} \sum_{\substack{x\in\Sigma^*:\\ \delta_\mathcal{A}(q0_\mathcal{A},x)=s\\ \delta_\mathcal{B}(q0_\mathcal{B},x)=s'}} \sum_{\substack{a\in\Sigma:\\ \delta^\bullet_\mathcal{A}(s,a)=q\\ \delta^\bullet_\mathcal{B}(s',a)=q'}} \pi_\mathcal{A}(xa).\,\pi_\mathcal{B}(xa)$$

Et enfin, en développant et réorganisant les termes, nous obtenons :

$$\eta^*_{qq'} = \sum_{s\in Q_\mathcal{A}} \sum_{s'\in Q_\mathcal{B}} \sum_{\substack{a\in\Sigma:\\ \delta^\bullet_\mathcal{A}(s,a)=q\\ \delta^\bullet_\mathcal{B}(s',a)=q'}} \sum_{\substack{x\in\Sigma^*:\\ \delta_\mathcal{A}(q0_\mathcal{A},x)=s\\ \delta_\mathcal{B}(q0_\mathcal{B},x)=s'}} \pi_\mathcal{A}(x).\,\pi_\mathcal{B}(x).\,\mathrm{p}^\bullet(s,a).\,\mathrm{p}^\bullet(s',a)$$

On reconnaît dans le terme central la formulation de $\eta_{ss'}$. Ainsi, nous avons une formule de récurrence pour calculer les paramètres η pour tous les couples d'états des automates.

$$\eta_{qq'} = \begin{cases} 1 + \displaystyle\sum_{s\in Q_\mathcal{A}} \sum_{s'\in Q_\mathcal{B}} \sum_{\substack{a\in\Sigma:\\ \delta^\bullet_\mathcal{A}(s,a)=q\\ \delta^\bullet_\mathcal{B}(s',a)=q'}} \eta_{ss'}.\,\mathrm{p}^\bullet(s,a).\,\mathrm{p}^\bullet(s',a) & \text{si } q = q0_\mathcal{A} \text{ et } q' = q0_\mathcal{B} \\ \displaystyle\sum_{s\in Q_\mathcal{A}} \sum_{s'\in Q_\mathcal{B}} \sum_{\substack{a\in\Sigma:\\ \delta^\bullet_\mathcal{A}(s,a)=q\\ \delta^\bullet_\mathcal{B}(s',a)=q'}} \eta_{ss'}.\,\mathrm{p}^\bullet(s,a).\,\mathrm{p}^\bullet(s',a) & \text{sinon} \end{cases}$$

Cette dernière formulation permet de représenter les $\eta_{qq'}$ comme les variables d'un système d'équations linéaires du premier degré de taille $t = |Q_\mathcal{A}| \times |Q_\mathcal{B}|$. Les méthodes formelles classiques de type « pivot de Gauss » et associées donnent des complexité calculatoire en $\mathcal{O}(t^3)$. Dans la pratique, la convergence des résultats est triviale si on remarque la monotone décroissance de la taille des mots considérés pour les calculs successifs, ceci nous permet d'obtenir des résultats suffisamment précis au bout de k étapes itératives de résolutions en fixant initialement les coefficients η à 0. La complexité de résolution passe alors à $\mathcal{O}(t \times k)$

Nous définissons maintenant une fonction qui mesure la probabilité pour que deux automates génèrent indépendemment une même chaîne. Cette grandeur, appelée probabilité de co-émission, est introduite sur une sous-classe des Hmm [LPN99], nous l'adaptons aux DPFA.

Définition 8.4 (Probabilité de co-émission) *Soient $\mathcal{A}, \mathcal{B}$, deux* DPFA *définis sur un unique alphabet Σ, nous définissons la* probabilité de co-émission *ainsi :*

$$\mathrm{CoEm}(\mathcal{A}, \mathcal{B}) = \sum_{w\in\Sigma^*} \big(\mathrm{p}_\mathcal{A}(w).\,\mathrm{p}_\mathcal{B}(w)\big)$$

Remarque : Cette dernière grandeur peut être facilement reliée aux coefficients η définis précédemment. En effet :

$$\begin{aligned}
\mathrm{CoEm}(\mathcal{A},\mathcal{B}) &= \sum_{w\in\Sigma^*}\big(\mathrm{p}_{\mathcal{A}}(w).\,\mathrm{p}_{\mathcal{B}}(w)\big)\\
&= \sum_{q\in Q_{\mathcal{A}}}\sum_{q'\in Q_{\mathcal{B}}}\sum_{\substack{w\in\Sigma^*:\\ \delta_{\mathcal{A}}(q0_{\mathcal{A}},w)=q\\ \delta_{\mathcal{B}}(q0_{\mathcal{B}},w)=q'}}\big(\mathrm{p}_{\mathcal{A}}(w).\,\mathrm{p}_{\mathcal{B}}(w)\big)\\
&= \sum_{q\in Q_{\mathcal{A}}}\sum_{q'\in Q_{\mathcal{B}}}\sum_{\substack{w\in\Sigma^*:\\ \delta_{\mathcal{A}}(q0_{\mathcal{A}},w)=q\\ \delta_{\mathcal{B}}(q0_{\mathcal{B}},w)=q'}}\big(\pi_{\mathcal{A}}(w).\,\mathrm{f}_{\mathcal{A}}(q).\,\pi_{\mathcal{B}}(w).\,\mathrm{f}_{\mathcal{B}}(q')\big)\\
&= \sum_{q\in Q_{\mathcal{A}}}\sum_{q'\in Q_{\mathcal{B}}}\big(\eta_{qq'}.\,\mathrm{f}_{\mathcal{A}}(q).\,\mathrm{f}_{\mathcal{B}}(q')\big)
\end{aligned}$$

Nous donnons ci-après un lemme dont nous nous servons pour le théorème suivant.

Lemme 8.1 (Inégalité de Cauchy-Schwarz pour les automates) *Soient $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, trois* DPFA *définis sur un unique alphabet Σ. L'*inégalité de Cauchy-Schwarz *s'écrit ainsi :*

$$\sum_{w\in\Sigma^*}\Big(\big(\mathrm{p}_{\mathcal{A}}(w)-\mathrm{p}_{\mathcal{C}}(w)\big).\big(\mathrm{p}_{\mathcal{C}}(w)-\mathrm{p}_{\mathcal{B}}(w)\big)\Big) \leq \sqrt{\sum_{w\in\Sigma^*}\big(\mathrm{p}_{\mathcal{A}}(w)-\mathrm{p}_{\mathcal{C}}(w)\big)^2}.\sqrt{\sum_{w\in\Sigma^*}\big(\mathrm{p}_{\mathcal{C}}(w)-\mathrm{p}_{\mathcal{B}}(w)\big)^2}$$

Preuve : Soit le polynome :

$$X(\lambda)=\sum_{w\in\Sigma^*}\Big(\big(\mathrm{p}_{\mathcal{A}}(w)-\mathrm{p}_{\mathcal{C}}(w)\big)+\lambda\big(\mathrm{p}_{\mathcal{C}}(w)-\mathrm{p}_{\mathcal{B}}(w)\big)\Big)^2$$

X s'écrit :

$$X(\lambda)=\sum_{w\in\Sigma^*}\big(\mathrm{p}_{\mathcal{A}}(w)-\mathrm{p}_{\mathcal{C}}(w)\big)^2+2.\lambda.\sum_{w\in\Sigma^*}\Big(\big(\mathrm{p}_{\mathcal{A}}(w)-\mathrm{p}_{\mathcal{C}}(w)\big).\big(\mathrm{p}_{\mathcal{C}}(w)-\mathrm{p}_{\mathcal{B}}(w)\big)\Big) +\lambda^2.\sum_{w\in\Sigma^*}\big(\mathrm{p}_{\mathcal{C}}(w)-\mathrm{p}_{\mathcal{B}}(w)\big)^2$$

C'est un polynôme en λ positif ou nul, son discriminant réduit est négatif ou nul. Ainsi :

$$\Big(\sum_{w\in\Sigma^*}\Big(\big(\mathrm{p}_{\mathcal{A}}(w)-\mathrm{p}_{\mathcal{C}}(w)\big).\big(\mathrm{p}_{\mathcal{C}}(w)-\mathrm{p}_{\mathcal{B}}(w)\big)\Big)\Big)^2 \leq \sum_{w\in\Sigma^*}\big(\mathrm{p}_{\mathcal{A}}(w)-\mathrm{p}_{\mathcal{C}}(w)\big)^2.\sum_{w\in\Sigma^*}\big(\mathrm{p}_{\mathcal{C}}(w)-\mathrm{p}_{\mathcal{B}}(w)\big)^2$$

Et donc, comme tout est positif,

$$\sum_{w\in\Sigma^*}\Big(\big(\mathrm{p}_{\mathcal{A}}(w)-\mathrm{p}_{\mathcal{C}}(w)\big).\big(\mathrm{p}_{\mathcal{C}}(w)-\mathrm{p}_{\mathcal{B}}(w)\big)\Big)$$

$$\leq \sqrt{\sum_{w\in\Sigma^*} \big(\mathrm{p}_{\mathcal{A}}(w) - \mathrm{p}_{\mathcal{C}}(w)\big)^2}.\sqrt{\sum_{w\in\Sigma^*} \big(\mathrm{p}_{\mathcal{C}}(w) - \mathrm{p}_{\mathcal{B}}(w)\big)^2}$$

Ce résultat nous sert dans la preuve de l'inégalité triangulaire des distances □

Nous disposons maintenant de tous les éléments pour développer le calcul de la distance euclidienne.

Théorème 8.1 *La mesure* d_2 *définie sur les distributions de probabilité est finie, est une vraie distance, et est calculable sur les* DPFA.

Preuve : Nous montrons ci-après les diverses propriétés du théorème.

1. Montrons que cette grandeur est toujours finie. Pour cela nous montrons facilement que la fonction de co-émission est finie.
   Par définition :

   $$\mathrm{CoEm}(\mathcal{A},\mathcal{B}) = \sum_{w\in\Sigma^*} \big(\mathrm{p}_{\mathcal{A}}(w).\,\mathrm{p}_{\mathcal{B}}(w)\big)$$

   De plus, $\forall w \in \Sigma^*$, $\mathrm{p}_{\mathcal{B}}(w) \leq 1$, ainsi :

   $$\mathrm{CoEm}(\mathcal{A},\mathcal{B}) \leq \sum_{w\in\Sigma^*} \big(\mathrm{p}_{\mathcal{A}}(w)\big) = 1$$

   La fonction de co-émission est donc toujours inférieure ou égale à 1, et est donc finie. La mesure d_2 est donc finie.
2. Pour montrer que cette mesure est une distance au sens mathématique du terme, nous donnons les propriétés d'une telle distance d, pour des éléments x,y,z de l'espace concerné.

   (a) $\mathrm{d}(x,y) \geq 0$
   (b) $\mathrm{d}(x,y) = 0 \Longleftrightarrow x = y$
   (c) $\mathrm{d}(x,y) = \mathrm{d}(y,x)$
   (d) $\mathrm{d}(x,y) \leq \mathrm{d}(x,z) + \mathrm{d}(z,y)$

   Reprenons point par point les propriétés pour la distance d_2 sur les DPFA. Nous considérons comme éléments $x = \mathcal{A}$, $y = \mathcal{B}$, $z = \mathcal{C}$ des distributions de probabilités représentées par des DPFA.

   (a) $\mathrm{d}_2(\mathcal{A},\mathcal{B}) \geq 0$: La somme de carrés dans $\mathbb{R}$ est trivialement positive ou nulle.
   (b) $\mathrm{d}_2(\mathcal{A},\mathcal{B}) = 0 \Longleftrightarrow \mathcal{A} = \mathcal{B}$: La somme de carrés dans $\mathbb{R}$ est nulle si et seulement si tous les termes $\mathrm{p}_{\mathcal{A}}(w) - \mathrm{p}_{\mathcal{B}}(w)$ sont nuls, c'est à dire si et seulement si $\mathrm{p}_{\mathcal{A}} = \mathrm{p}_{\mathcal{B}}$. Les distributions $\mathcal{A}$ et $\mathcal{B}$ sont donc égales.
   (c) $\mathrm{d}_2(\mathcal{A},\mathcal{B}) = \mathrm{d}_2(\mathcal{B},\mathcal{A})$: trivial.

(d) $\mathrm{d}_2(\mathcal{A},\mathcal{B}) \leq \mathrm{d}_2(\mathcal{A},\mathcal{C}) + \mathrm{d}_2(\mathcal{C},\mathcal{B})$:
Par définition,

$$\mathrm{d}_2(\mathcal{A},\mathcal{B}) = \sqrt{\sum_{w\in\Sigma^*} \big(\mathrm{p}_{\mathcal{A}}(w) - \mathrm{p}_{\mathcal{B}}(w)\big)^2}$$

Ajoutons artificiellement $\mathcal{C}$

$$\mathrm{d}_2(\mathcal{A},\mathcal{B}) = \sqrt{\sum_{w\in\Sigma^*} \Big(\big(\mathrm{p}_{\mathcal{A}}(w) - \mathrm{p}_{\mathcal{C}}(w)\big) + \big(\mathrm{p}_{\mathcal{C}}(w) - \mathrm{p}_{\mathcal{B}}(w)\big)\Big)^2}$$

On pose $M = \mathrm{p}_{\mathcal{A}}(w) - \mathrm{p}_{\mathcal{C}}(w)$ et $N = \mathrm{p}_{\mathcal{C}}(w) - \mathrm{p}_{\mathcal{B}}(w)$
Développons

$$\mathrm{d}_2(\mathcal{A},\mathcal{B}) = \sqrt{\sum_{w\in\Sigma^*} M^2 + \sum_{w\in\Sigma^*} N^2 + 2\sum_{w\in\Sigma^*} (M.N)}$$

Appliquons l'inégalité de Cauchy-Schwartz

$$\mathrm{d}_2(\mathcal{A},\mathcal{B}) \leq \sqrt{\sum_{w\in\Sigma^*} M^2 + \sum_{w\in\Sigma^*} N^2 + 2\sqrt{\sum_{w\in\Sigma^*} M^2}.\sqrt{\sum_{w\in\Sigma^*} N^2}}$$

Factorisons

$$\mathrm{d}_2(\mathcal{A},\mathcal{B}) \leq \sqrt{\left(\sqrt{\sum_{w\in\Sigma^*} M^2} + \sqrt{\sum_{w\in\Sigma^*} N^2}\right)^2}$$

$$\mathrm{d}_2(\mathcal{A},\mathcal{B}) \leq \mathrm{d}_2(\mathcal{A},\mathcal{C}) + \mathrm{d}_2(\mathcal{C},\mathcal{B})$$

3. La calculabilité de la distance tient dans la résolution du système linéaire des coefficients η.

□

Il est intéressant d'utiliser la distance d_2 pour de multiples raisons. Tout d'abord, comparée à la d_{KL}, la d_2 ne souffre pas des maux de la différence d'entropie. La distance est toujours finie, quel que soit le modèle appris, le lissage n'est donc pas important. Ensuite, il est possible d'utiliser cette mesure hors du contexte d'apprentissage comme classifieur.

Nous verrons dans la section suivante diverses expérimentations menées pour évaluer le comportement pratique de cette mesure, aussi bien en apprentissage — où elle est comparée à la d_{KL} — qu'en classification.

8.4 Expérimentations ⋆

Les diverses expérimentations expliquées ci-après ont été menées pour voir comment se comporte cette mesure dans différentes tâches. En apprentissage, nous verrons comment évaluer les résultats, dans le cas d'identification stricte où le fait de trouver la cible correspond à avoir une d_2 minimale, et sur une tâche pratique de la reconnaissance des formes, nous verrons que l'évaluation est similaire si l'on utilise la d_{KL} sur un modèle qu'il faut lisser ou la d_2 directement sur le modèle appris. Deux autres expérimentations viennent compléter la phase de test du comportement de cette distance : une montrant la vitesse de convergence de la distance, et donc sa robustesse à l'échantillonnage — ce qui nous permet de définir depuis quelle population est tirée un échantillon parmi plusieurs populations possibles — l'autre insistant sur les qualités de classification, dans cette dernière nous essaierons de connaître l'auteur d'un poème en ayant appris certains recueils précédemment.

8.4.1 Comportement dans le contexte de l'identification exacte ⋆

Pour estimer le comportement de la d_2 sur des modèles appris dans le cadre d'une identification, nous apprenons des modèles stochastiques que nous comparons à la cible connue. Nous avons choisi de sélectionner la grammaire de Reber qui a été initialement [Reb67] créée pour prouver que l'apprentissage humain est implicite (c.-à-d. les règles apprises et utilisées ne sont pas conscientes).

Nous créons une cible ayant pour structure la grammaire de Reber à laquelle nous ajoutons des probabilités de transition pour faire un modèle de langage sous la forme d'un DPFA. La figure 8.1 montre cette distribution sous la forme d'un automate. De là,

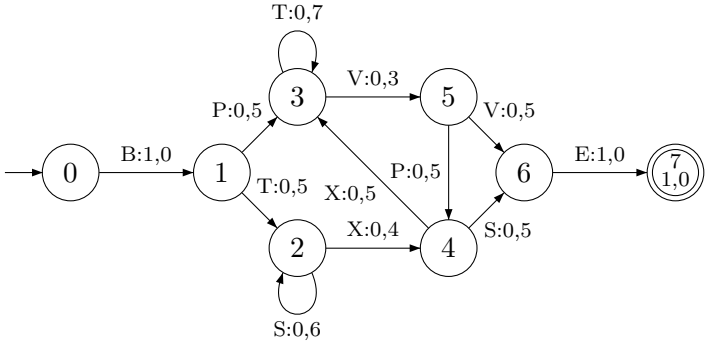


Fig. 8.1 – *La grammaire de* Reber

nous générons en suivant cette distribution trois échantillons de mots de taille 100, 500 et 1000. Nous utilisons ensuite l'algorithme Alergia présenté dans le chapitre 6, sur lequel nous faisons varier le paramètre de généralisation α pour trouver la cible. Plutôt que la d_{KL}, calculable quand la cible est connue, nous utilisons la perplexité pour des raisons pratiques. Pour ne pas pénaliser l'interprétation de cette dernière mesure, nous choisissons un lissage dit « par repli » qui donne de très bon résultats pratiques sur les modèles appris sur ces données. Nous calculons aussi la d_2 entre l'automate appris (non

lissé) et le modèle cible.

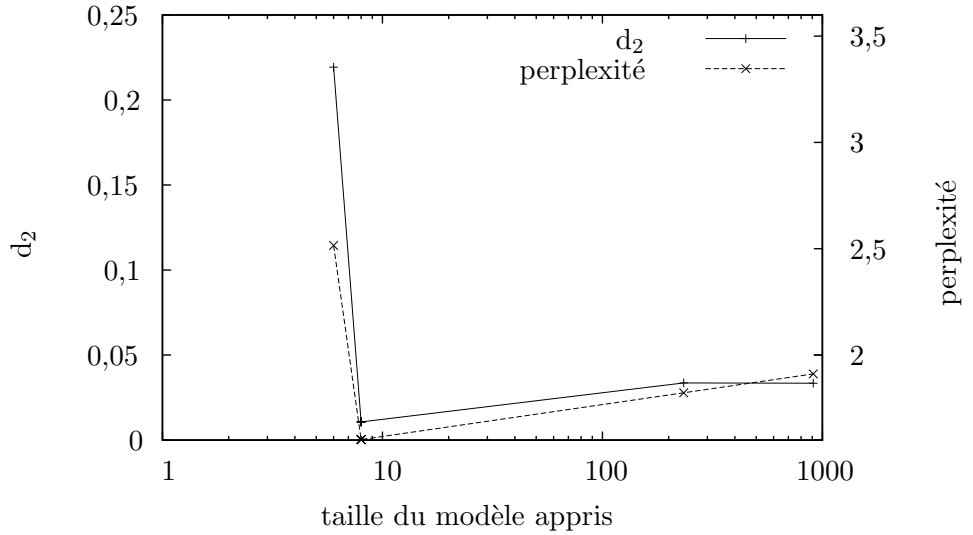


FIG. 8.2 – *Comparaison entre la perplexité et la distance* d_2 *sur la grammaire de* REBER

La figure 8.2 montre sur le même graphique le comportement de la d_{KL} (perplexité) et de la d_2 en fonction de la taille de l'automate appris. La cible, la grammaire de REBER, compte huit états. L'abscisse du graphique représente la taille de l'automate appris en nombre d'états. Ce nombre varie en fonction de la permissivité laissée par le paramètre α de l'algorithme utilisé. La zone intéressante se situe où les automates sont proches de la taille de la cible. En cet endroit, le point de calcul pour la d_2, comme pour la perplexité, correspond en fait à cinq automates appris pour α dans l'intervalle $[0{,}005; 0{,}08]$. Ces courbes sont données pour un échantillon d'apprentissage de taille 500.

Nous remarquons facilement sur la figure que la d_2 comme la d_{KL} sont minimales ensembles pour une taille d'automate de huit états. Les cinq automates appris correspondant à ce point ont bien la structure de la cible, seule l'estimation des probabilités des transitions est légèrement différente. Les résultats sur les échantillons de taille 100 et 1000 montrent que cette estimation devient meilleure avec l'augmentation du nombre d'exemples. Les automates plus petits ou plus grands — qui ne correspondent donc pas à la structure de la cible — ont des mesures d'éloignement plus grandes. La d_2, comme la d_{KL} dans ce contexte, est donc discriminante pour évaluer ces modèles appris. De plus, ne pas être obligé de lisser le modèle appris est un avantage certain pour l'utilisation de la distance euclidienne plutôt que la différence d'entropie relative.

Cette expérimentation met en avant les résultats sur un apprentissage de cible connue avec des données artificiellement générées par nos soins. Le principal intérêt de l'apprentissage automatique est de modéliser par un langage un phénomène naturel. L'exemple de la reconnaissance vocale est souvent avancé dans ce contexte, c'est le sujet

de l'expérimentation suivante.

8.4.2 Comparaison d_{KL}-d_2 sur Atis ⋆

L'expérimentation menée est similaire à la précédente mais, se basant sur des données réelles, la cible n'est donc pas connue. Nous voulons néanmoins évaluer la qualité du modèle appris dans le cadre de la reconnaissance de la parole. Les données utilisées sont des phrases dites par des personnes réservant un billet d'avion sur un serveur vocal aux États-Unis. Cette base en connue sous le nom d'Atis [Hir92]. Chaque exemple est une phrase correspondant à une requête de l'utilisateur : réservation avec lieux de départ, d'arrivée, heures désirées ; ou demande de renseignements concernant les lieux, les horaires, des différents vols disponibles. La table 8.1 montre quelques phrases extraites de la base utilisée.

FIND THE CHEAPEST FLIGHT FROM ATLANTA TO PHILADELPHIA
ROUND TRIP
WHAT TYPE OF AIRCRAFT IS USED FROM BOSTON TO PHILADELPHIA
BEFORE TWELVE NOON
I'D LIKE TO FIND THE CHEAPEST ONE WAY FARE FROM BOSTON TO
PITTSBURGH AND THEN BACK FROM PITTSBURGH TO BOSTON
FIND A FLIGHT FROM ATLANTA TO BALTIMORE TO ARRIVE AROUND
SEVEN P M

Tab. 8.1 – *Exemples de phrases de la base* Atis

Dans le cas où la cible est inconnue, nous désirons apprendre un modèle de langage se conformant le plus possible aux phrases d'un échantillon d'apprentissage. La base Atis est découpée en trois ensembles distincts : un ensemble d'apprentissage, un ensemble dit « de développement », et un ensemble de test pour pouvoir évaluer les modèles appris. Les ensembles ont des tailles respectives de 13044, 974 et 1001 phrases. Le protocole mis en place se définit ainsi : tout d'abord, nous apprenons plusieurs modèles de langages avec l'algorithme Alergia en faisant varier le paramètre α, puis chaque modèle ainsi obtenu est comparé à la distribution statistique des exemples de l'ensemble de test. Cette comparaison est faite soit grâce à la perplexité — le modèle est alors lissé, soit en calculant la d_2 entre le modèle appris et le PPTA de l'ensemble de test — le PPTA représente exactement la distribution des exemples de l'ensemble sur lequel il est construit.

La figure 8.3 rassemble les résultats d'évaluation des modèles appris avec la perplexité d'une part et la distance euclidienne d'autre part. De même que précédemment, l'abscisse du graphique représente la taille de l'automate appris en nombre d'états. Les résultats sont donnés pour des automates appris sur l'ensemble complet d'apprentissage, soit 13044 phrases. Les modèles appris ont de 133 à 73412 états suivant la valeur de α comprise entre 0,00001 et 1.

Les résultats sont intéressants dans le sens où l'automate ayant la plus faible perplexité n'est pas exactement celui ayant la plus faible distance. La modélisation du

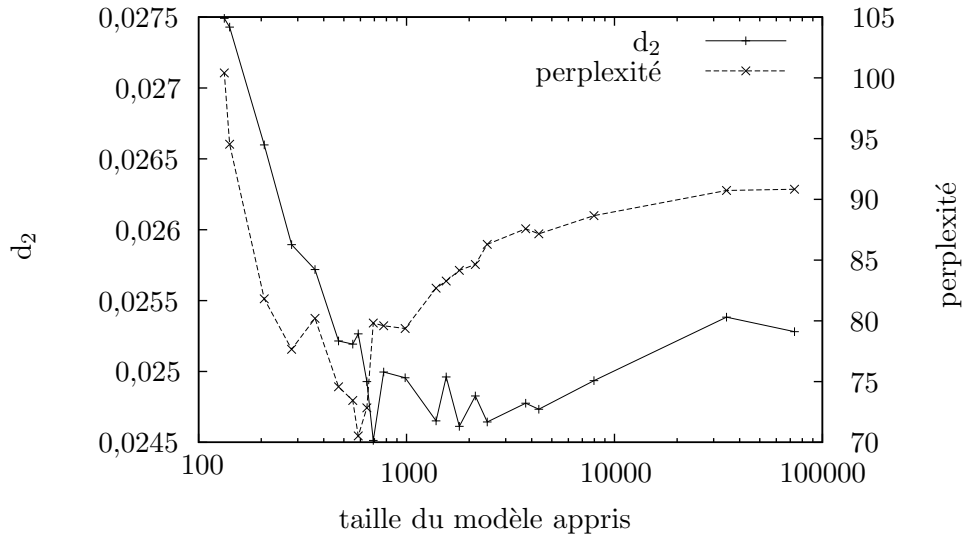


FIG. 8.3 – *Comparaison entre la perplexité et la distance* d_2 *sur la base* ATIS

langage par un langage régulier peut être vue comme une approximation, le modèle de langage cible, non connu, est sûrement non-régulier et ne peut donc qu'être approximé par ces méthodes. Néanmoins, de manière globale, les comportements des deux mesures sont similaires : lorsque l'automate appris possède un grand nombre d'états, il y a un phénomène de sous-apprentissage et le modèle obtenu est médiocre. À l'inverse, lorsque l'automate possède trop peu d'états, il n'est pas assez discriminant, il y a sur-apprentissage et le modèle résultant est tout aussi médiocre. Seule la zone de transition, où les deux mesures sont très faibles, renferme de bons candidats pour la modélisation de ces phrases parlées. De plus, le modèle est obligatoirement lissé pour le calcul de la perplexité, ce qui peut le rapprocher sensiblement de la distribution cherchée initialement. Les résultats obtenus pour la distance ont l'avantage de n'évaluer que l'apprentissage et non le couple apprentissage-lissage.

Les deux expérimentations précédentes montrent que le comportement de la d_2 pour évaluer les modèles appris est très intéressant : en effet, son comportement est similaire à celui de la perplexité qui est la mesure la plus souvent utilisée dans l'état de l'art et, de plus, qui ne nécessite pas de lisser l'automate appris.

Un autre avantage de la distance euclidienne est son pouvoir discriminant que nous mettrons en avant en classification, et la possibilité de choisir une distribution origine, parmi plusieurs possibles, à un échantillon d'exemples, ce que nous montrons dans l'expérimentation suivante.

8.4.3 De la robustesse à l'échantillonnage ⋆

À présent nous dépassons les limites du seul contexte de l'apprentissage, afin de montrer le pouvoir discriminant et la facilité d'utilisation de la distance d'automates lorsque l'on souhaite établir la provenance d'un échantillon par rapport à plusieurs distributions sources possibles. Cette tâche est très importante en détection ou en prédiction dans le cas où les modèles sont très proches les uns des autres et difficilement différentiables.

L'exemple que nous avons choisi illustre parfaitement ce cas de figure : nous allons établir la provenance d'un échantillon stochastique tiré depuis une distribution représentée par une fonction de parité. Ces entités sont souvent présentées [KV89] comme très difficiles à apprendre de part leur possible similitude.

Nous définissons tout d'abord ce que nous appelons fonction de parité, puis nous explicitons le protocole d'expérimentation. Considérons tout d'abord, $\Sigma = \{a,b\}$ un alphabet de deux lettres. Une fonction de parité de taille n peut se représenter sous la forme d'une structure de DPFA à $2n+1$ états. Cet automate accepte un mot $w \in \Sigma^*$ si et seulement si :

- $|w| = n$ et le nombre de b dans certaines positions prédéfinies est pair ;
- $|w| = n+1$ et le nombre de b dans certaines positions prédéfinies est impair.

Ces positions sont appelées *positions d'inversion.* Une chaîne binaire peut aussi représenter une telle fonction. Par exemple, la chaîne 101101 représente la fonction de parité illustrée dans la figure 8.4. Pour des raisons de clarté, les probabilités ne sont pas représentées. La table 8.2 rassemble quelques exemples d'acceptation ou non de mots par

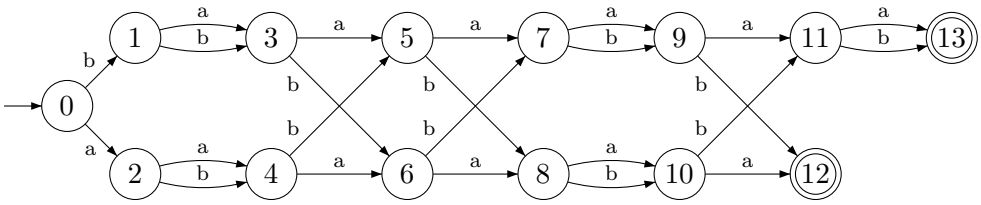


Fig. 8.4 – *Représentation de la fonction de parité* 101101 *par un* DPFA

la fonction de parité 101101.

L'expérimentation a pour but de comparer des échantillons issus de trois distributions différentes représentées pour la structure par des fonctions de probabilité, $f_1 = 101101$ avec des probabilités de 0,5 sur chaque transition, $f_2 = 011010$ avec des probabilités de 0,5 sur chaque transition et f_3 qui partage la même représentation structurelle que f_1, mais avec des probabilités de transitions différentes.

Pour estimer de plus la vitesse de convergence de discrimination, nous utilisons différentes tailles pour la comparaison des échantillons de 10 à 100000 mots. Pour chaque modèle f_i, nous générons suivant la distribution quinze échantillons $\{s_{i_1},s_{i_2},\ldots,s_{i_{15}}\}$. Nous calculons ensuite chaque $\mathrm{d}_2^\bullet(s_i,f_j)$, moyenne arithmétique des distances entre les

Mot soumis	1	0	1	1	0	1		Parité	Taille	Acceptation
aaaaaa	*a*	*a*	*a*	*a*	*a*	*a*		pair	6	✔
aaaaaaa	*a*	*a*	*a*	*a*	*a*	*a*	*a*	pair	7	✘
ababbaa	*a*	*b*	*a*	*b*	*b*	*a*	*a*	impair	7	✔
abbaba	*a*	*b*	*b*	*a*	*b*	*a*		impair	6	✘
abbab	*a*	*b*	*b*	*a*	*b*			impair	5	✘

TAB. 8.2 – *Exemples d'analyse de mots par un* DPFA *représentant la fonction* 101101 *de taille* 6

échantillons issus du modèle f_i et le modèle f_j complet. Encore une fois, le modèle pris en compte pour représenter les échantillons est le PPTA correspondant.

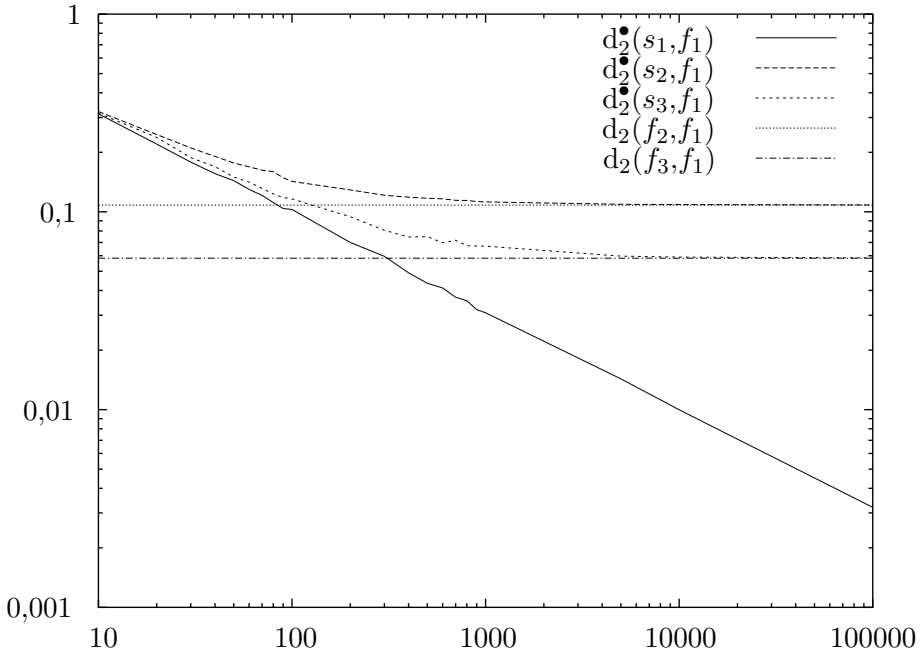


FIG. 8.5 – *Pouvoir discriminant et convergence de la* d_2

La figure 8.5 montre les résultats obtenus en prenant le modèle f_1 comme référence. Les trois premières courbes donnent la moyenne des distances des échantillons s_1, s_2 et s_3 par rapport au modèle f_1, tandis que les deux dernières donnent la distance d_2 réelle entre les modèles f_2,f_1 et f_3,f_1. La comparaison avec les modèles f_2 ou f_3 comme référence donne des résultats équivalents.

L'ordonnée du graphique représente la valeur des distances, et l'abscisse la taille de l'échantillon traité dans les cas des $d_2^\bullet$. Les deux échelles sont logarithmiques.

Dans un premier temps, nous pouvons affirmer qu'empiriquement chacune des distances entre un échantillon s_2 ou s_3 et le modèle f_1 converge rapidement vers la valeur

asymptotique de la distance du modèle issu du modèle f_1. De plus, nous notons que $\mathrm{d}_2^{\bullet}(s_1,f_1)$ converge elle vers 0. Nous notons par ailleurs que la vitesse de convergence est rapide : sur la figure une taille d'échantillon de $200-300$ nous permet déjà de trancher sur l'origine de l'échantillon.

Enfin, nous remarquons quelque chose de très intéressant, à taille d'échantillon égale, les échantillons de s_3 sont toujours plus près du modèle f_1 que les échantillons s_2 : en rappelant que f_1 et f_3 partagent la même structure, et sont donc très proches, nous confortons encore le bon comportement de la distance d_2.

8.4.4 Classification de poèmes ⋆

Le pouvoir de discrimination de la d_2 et son utilisation immédiate — sans problème de lissage, permet d'envisager de l'utiliser pour de nombreuses tâches en classification. Depuis quelques années, quelques grands spécialistes linguistiques se penchent sur l'éventualité que Corneille puisse avoir écrit des œuvres de Molière [Lab03]. Loin de vouloir rentrer dans le débat à l'heure actuelle, nous proposons simplement dans cette partie d'apprendre des modèles de langages sur des recueils de poèmes de Lamartine et Hugo, et de classer en aveugle des poèmes d'autres recueils des mêmes auteurs.

Le protocole expérimental est le suivant : nous disposons de quatre recueils de poèmes, deux de chaque auteurs (cf. tables 8.3 et 8.4 pour une description succincte des données). Nous définissons comme unité lexicale un vers de chaque poème, c.-à-d. que les éléments de Σ^* sont les vers, et les éléments de l'alphabet Σ sont les mots. Nous apprenons sur chacun des recueils un modèle de langage en fixant simplement le paramètre α de l'algorithme Alergia, la qualité de l'apprentissage n'est pas le point que nous voulons mettre en avant. Nous notons M_1, M_2 et H_1, H_2 les modèles appris sur les premiers et seconds recueils de poèmes de, respectivement Lamartine et Hugo. De la même manière, nous notons M_i^j et H_i^j les j^{es}poèmes des i^{es}recueils des auteurs correspondants.

Nous calculons ensuite la distance entre les poèmes de la façon suivante : les couples représentent (table 8.5) les distances comparées pour classifier les poèmes. Ainsi, nous comptabilisons comme un succès de classification le fait que la distance par rapport au modèle du même auteur est plus faible que celle par rapport au modèle de l'autre auteur. Nous en déduisons ensuite un taux de succès en classification. C'est ce taux qui est présenté en table 8.6. La première colonne de la table 8.6 donne le recueil où les poèmes sont choisis pour les comparer suivant les règles présentées dans la table 8.5. Nous pouvons voir que les résultats en classification sont désastreux pour les collections H_i : un tirage aléatoire aurait été meilleur. Pourquoi ? La distance d_2 mesure l'éloignement entre les modèles de langages définis sur les vers des poèmes. Une analyse approfondie montre que très peu de vers entiers sont employés plusieurs fois par le même auteur. Rappelons que la distance euclidienne se définit suivant les probabilités de co-émission de chaînes du langage, sur $(\mathcal{A},\mathcal{A})$, $(\mathcal{B},\mathcal{B})$ ou $(\mathcal{A},\mathcal{B})$. Or, la co-émission conjointe sur $(\mathcal{A},\mathcal{B})$ est ici très proche de zéro, la classification repose donc uniquement sur le fait qu'une des deux co-émissions $(\mathcal{A},\mathcal{A})$ ou $(\mathcal{B},\mathcal{B})$ est plus grande que l'autre, ce qui

	Lamartine	Hugo
Recueil	Harmonies poétiques et religieuses	Les contemplations I
Titre	Aux chrétiens dans les temps d'épreuves	La coccinelle
Extrait	Pourquoi vous troublez-vous, enfants de l'Evangile? À quoi sert dans les cieux ton tonnerre inutile, Disent-ils au Seigneur, quand ton Christ insulté, Comme au jour où sa mort fit trembler les collines, Un roseau dans les mains et le front ceint d'épines, Au siècle est présenté?	Elle me dit: -Quelque chose -Me tourmente.- Et j'aperçus Son cou de neige, et dessus, Un petit insecte rose.
Recueil	Méditations poétiques	Les contemplations II
Titre	Le soir	Au fils d'un poëte
Extrait	Le soir ramène le silence. Assis sur ces rochers déserts, Je suis dans le vague des airs Le char de la nuit qui s'avance.	Enfant, laisse aux mers inquiètes Le naufragé, tribun ou roi; Laisse s'en aller les poëtes ! La poésie est près de toi.

Tab. 8.3 – *Exemples de poèmes*

implique un classement de la quasi-totalité des exemples vers ce modèle.

L'unité lexicale est sûrement mal choisie : une analyse linguistique pourrait peut-être guider le choix de la bonne unité de base. Deux poèmes d'un même auteur, hormis le vocabulaire (les éléments de Σ), doivent avoir des constructions grammaticales en commun. En considérant le calcul de la distance euclidienne, nous nous apercevons que nous disposons aussi d'une fonction de probabilité préfixielle. L'idée est donc de calculer une mesure, similaire à la d_2, avec la fonction de probabilité préfixielle à la place de la probabilité du mot (ici le vers) lui même. Nous définissons formellement cette mesure d'éloignement.

Définition 8.5 (Distance euclidienne préfixielle) *Soient deux distributions de probabilité $\mathfrak{D}$ et $\mathfrak{D}'$, nous définissons la* distance euclidienne préfixielle *entre ces deux entités comme :*

$$\mathrm{d}_{2p}(\mathfrak{D},\mathfrak{D}') = \sqrt{\sum_{w\in\Sigma^*} \left(\pi_{\mathfrak{D}}(w) - \pi_{\mathfrak{D}'}(w)\right)^2}$$

Définition 8.6 (Probabilité de co-émission préfixielle) *Soit Σ un alphabet fini,*

Recueil	Nb. poèmes	Nb. vers	Nb. mots
H_1	87	2381	46686
H_2	68	3067	61322
M_1	15	1079	27243
M_2	30	1398	26483

Tab. 8.4 – *Informations sur les données de classification*

comparer à	
$\mathrm{d}_2(M_1^j,M_2)$	$\mathrm{d}_2(M_1^j,H_2)$
$\mathrm{d}_2(M_2^j,M_1)$	$\mathrm{d}_2(M_2^j,H_1)$
$\mathrm{d}_2(H_1^j,H_2)$	$\mathrm{d}_2(H_1^j,M_2)$
$\mathrm{d}_2(H_2^j,H_1)$	$\mathrm{d}_2(H_2^j,M_1)$

Tab. 8.5 – *Couples de distances calculées pour classifier les poèmes*

soient $\mathcal{A},\mathcal{B}$*, deux* DPFA *définis sur cet unique alphabet, nous définissons la* probabilité de co-émission préfixielle *ainsi :*

$$\mathrm{CoEmPref}(\mathcal{A},\mathcal{B}) = \sum_{w\in\Sigma^*} \big(\pi_{\mathcal{A}}(w).\,\pi_{\mathcal{B}}(w)\big)$$

Remarque : Nous pouvons remarquer que cette dernière grandeur est directement liée aux coefficients η définis pour le calcul de la d_2. En effet :

$$\begin{aligned}
\mathrm{CoEmPref}(\mathcal{A},\mathcal{B}) &= \sum_{w\in\Sigma^*} \big(\pi_{\mathcal{A}}(w).\,\pi_{\mathcal{B}}(w)\big) \\
&= \sum_{q\in Q_{\mathcal{A}}} \sum_{q'\in Q_{\mathcal{B}}} \sum_{\substack{w\in\Sigma^*:\\ \delta_{\mathcal{A}}(q0_{\mathcal{A}},w)=q\\ \delta_{\mathcal{B}}(q0_{\mathcal{B}},w)=q'}} \big(\pi_{\mathcal{A}}(w).\,\pi_{\mathcal{B}}(w)\big) \\
&= \sum_{q\in Q_{\mathcal{A}}} \sum_{q'\in Q_{\mathcal{B}}} \big(\eta_{qq'}\big)
\end{aligned}$$

En reprenant les travaux menés sur la distance euclidienne d_2, nous énonçons un théorème similaire.

Théorème 8.2 *La mesure* d_{2p} *définie sur les distributions de probabilité est finie, est une vraie distance, et est calculable sur les* DPFA.

Preuve : Les preuves des différents points sont développées en annexe 2. De la même façon que pour la d_2, les résultats suivent si l'on remarque que l'on peut réécrire cette mesure sous la forme suivante :

$$\mathrm{d}_{2p}(\mathcal{A},\mathcal{B}) = \sqrt{\mathrm{CoEmPref}(\mathcal{A},\mathcal{A}) + \mathrm{CoEmPref}(\mathcal{B},\mathcal{B}) - 2.\,\mathrm{CoEmPref}(\mathcal{A},\mathcal{B})}$$

□

Recueil de base	Taux de succès de la d_2
H_1	2,30 %
H_2	8,82 %
M_1	100,00 %
M_2	100,00 %

TAB. 8.6 – *Taux de succès en classification avec la* d_2 *- unité lexicale : le vers*

L'utilisation de la langue par les poètes nous fait penser que les facteurs des mots (sous-chaînes) seraient une bonne unité lexicale de base. Malheureusement, aucun développement ne nous permet de calculer une distance directement sur les facteurs contrairement aux préfixes. De manière pratique, nous estimons une mesure sur les facteurs, par le calcul de la d_{2p} sur l'ensemble des suffixes des vers des poèmes. Le protocole expérimental reste inchangé, les données sont les mêmes, seuls changent les calculs de distances. Les résultats sont données dans la table 8.7. Les résultats obtenus ainsi sur-

Recueil de base	Taux de succès	
	d_{2p}	d_{2p} sur les suffixes
H_1	70,1 %	81,6 %
H_2	75,0 %	89,4 %
M_1	73,3 %	73,3 %
M_2	53,3 %	93,3 %

TAB. 8.7 – *Taux de succès en classification avec la* d_{2p} *- unité lexicale : les préfixes et facteurs des vers*

passent de beaucoup les précédents. Cela appuie l'hypothèse que les vers entiers ne sont pas représentatifs de l'auteur, mais les constructions internes, l'assemblage des mots deux par deux, trois par trois, *etc.*, est propre à chacun. La différence est cependant moins marquée entre les préfixes et les facteurs.

Par conséquent, nous montrons empiriquement, sur les données artificielles et réelles, que la distance euclidienne de modèles de langage est intéressante, et ceci aussi bien pour les fonctions de parité, qu'en classification de texte. De plus, le fait que la distance euclidienne ne nécessite aucun lissage des modèles la rend facilement utilisable.

8.5 Conclusion ⋆

Nous avons défini dans ce chapitre une nouvelle mesure entre modèles de langages. Nous en avons donné des résultats théoriques en terme de distance mathématique et observé son comportement intéressant dans les expérimentations. Cette distance permet l'évaluation des modèles appris sans qu'un lissage perturbateur ne soit nécessaire. Son pouvoir discriminant et sa vitesse de convergence élevée la rendent parfaitement adaptée à la tâche de classification.

Ce chapitre concernant les distances conclut la partie consacrée à l'inférence grammaticale, dans la suite nous présentons des séries d'expérimentations où nous nous servons des différents méthodes développées dans les parties extractions de données web et inférence grammaticale.

Troisième partie

Applications aux données web ⋆

Les travaux présentés dans cette partie ont donné lieu à deux communications en conférences internationales :

[MJ05] T. Murgue et P. Jaillon, *Data Preparation and Structural Models for Web Usage Mining*, Sciences of Electronic, Technologies of Information and Telecommunications (SETIT'05) (Sousse, Tunisie), 2005.

[Mur05b] T. Murgue, *Log Pre-Processing and Grammatical Inference for Web Usage Mining*, Workshop on Machine Learning for User Modeling: Challenges – UM 2005, 2005.

No amount of experimentation can ever prove me right; a single experiment can prove me wrong.
A. Einstein

9 Présentation ⋆

Sommaire

Résumé

Les données web sont, comme nous l'avons vu, difficile à obtenir. Les jeux de données que nous utilisons sont décrits de manière approfondie : origine, taille, morphologie. Nous notons certains points de différence entre les jeux de données.

9.1 Introduction ⋆

Les précédentes parties du manuscrit décrivent les données du *Web* et l'apprentissage automatique. Nous présentons les résultats [Mur05b, MJ05] d'expérimentations que nous avons menées dans chacun de ces thèmes : de l'évaluation morphologique[1] de la reconstruction des logs à l'évalution de l'apprentissage *via* une distance entre modèles de langages. Nous désirons maintenant montrer des expérimentations qui englobent les deux aspects : traitement préalable des données et inference grammaticale. Les données à utiliser, les protocoles d'expérimentation, l'évaluation des résultats, les résultats et leurs interprétations sont donnés dans cette partie.

Dans ce chapitre nous nous attachons uniquement à décrire les données que nous utiliserons. Deux types existent : les données artificielles et celles obtenus dans le monde réel. Nous donnons pour chacun de ces types de données :

- leur origine : c'est à dire le protocole de génération dans le cas de données artificielles, ou une description courte du site web de provenance ;
- le nombre de *logs* mis en œuvre dans l'expérimentation : ce nombre correspond à la taille des données d'apprentissage ;

[1] Nous ne nous sommes intéressés dans la section 3.3 qu'à la structure de logs reconstruits par rapport à des logs idéaux générés artificiellement

– pour les données réelles uniquement, la structure du site web et sa taille (en nombre de nœuds du graphe extrait correspondant).

Ainsi, avec ce type d'informations, nous allons comparer les résultats obtenus dans nos expérimentations à ceux connus dans la littérature.

9.2 Description des données artificielles ⋆

Les données artificielles que nous utilisons sont celles déjà présentées en partie dans la section 3.3. Nous utilisons comme structure de site web connu, les sites de l'équipe EURISE[2] [Eur02] et de l'association AGORA21[3] [Ago02]. Dans l'évaluation stricte de la reconstruction des logs, nous comparons la morphologie des visites détectées par notre méthode, avec ou sans la reconstruction, et les visites de référence, qui sont les chemins générés artificiellement. Maintenant, nous allons valider expérimentalement cette étape en montrant qu'elle améliore l'apprentissage en utilisant les critères d'évaluation présentés dans le chapitre 7. Pour obtenir les différents jeux de données à comparer, nous générons, dans un certain laps de temps, des visites utilisateur en sélectionnant un nœud dans le graphe du site web et en établissant un plus court chemin que nous bruitons volontairement pour ressembler un peu plus à une navigation réelle. En effet pour trouver une information d'une page précise, les utilisateurs ne passent pas forcément par le chemin le plus court et ont souvent recours aux retours. Nous modèlisons ces deux possibilités en insérant dans le calcul du chemin des probabilités de ne pas prendre la route la plus courte et une probabilité de revenir en arrière. Ces probabilités sont fixées à l'avance. Nous présentons dans la table 9.1, les diverses caractéristiques des données générées. Nous pouvons constater que les tailles des sites web considérés

	EURISE	AGORA21
Nb. de pages	105	4850
Nb. de chemins générés	1000	1000
Nb. visites de référence	148	137

TAB. 9.1 – *Description des données générées sur les sites* EURISE *et* AGORA21

diffèrent sensiblement en terme de nombres de pages.

Ces données nous serviront dans une expérimentation visant à montrer que les modèles appris sur les *logs* reconstruits sont meilleurs en terme d'écart à un échantillon de test (calculs avec la d_{KL} et la d_2). Des modèles appris sur les données du site EURISE nous serviront à prédire la page suivante dans la navigation ainsi qu'à montrer que sur les *logs* reconstruits, la prédiction est meilleure.

2 Site de l'Équipe Universitaire de Recherche en Informatique de Saint-Étienne

3 Site web de l'association Agora21 traitant du développement durable. Depuis, cette association a fusionné avec le « pôle de l'eau » et « l'Association pour les Pratiques du Développement Durable » pour former le CIRIDD [Cir]

9.3 Description des données réelles ⋆

Pour tester nos méthodes de reconstruction et d'évaluation en contexte réel, nous disposons de deux jeux de données correspondant aux *logs* de deux sites web [Eur02, MRI02], ainsi qu'à leur structure complète (extraite depuis les pages des sites en question). Pour les deux sites, les données sont récupérées sour forme de fichiers de *logs* et d'arborescence de fichiers représentant les pages du sites web. Les fichiers d'enregistrements des transactions contiennent une grande quantité de données. Nous résumons les différentes caractéristiques dont nous disposons sur ces jeux de données dans la table 9.2. Nous pouvons remarquer là encore que les sites diffèrent en terme de nombre

	Eurise	MRIM
Nb. de pages	105	6329
Nb. de *hits*	180442	827308
Nb. visites de référence	5869	10786
Taille moyenne des visistes (pages vues)	3,67	3,28
Taille de la visite la plus grande	188	402
Nb. visites détectées par notre méthode	5219	36405
Taille moyenne des visistes (pages vues)	5,67	10,44
Taille de la visite la plus grande	704	10528

Tab. 9.2 – *Description des données réelles correspondant aux sites* Eurise *et* MRIM

de pages. De plus, suivant la méthode utilisée — les visites de référence sont extraites sans la reconstruction des *logs* alors que celles données en deuxième partie de tableau sont extraites avec la reconstruction — les tailles moyennes des visites diffèrent grandement. La taille de la visite la plus grande est peu importante mais permet de montrer que les méthodes employées ne donnent pas de résultats parfaits : en effet, une visite de plus de dix mille pages paraît peu probable.

Ces données seront utilisées dans une tâche de prédiction de la page suivante dans une navigation utilisateur. Cette tâche est le point de démarrage classique pour l'adaptation de site web : lorsqu'on peut connaître la page que va demander l'utilisateur, il est possible de lui proposer cette page à l'avance pour améliorer le service.

9.4 Conclusion ⋆

Dans ce chapitre nous avons défini de manière appronfondie la morphologie des données que nous utilisons dans deux types d'expérimentations : une sur l'évaluation des modèles appris par calcul de l'écart à une distribution sur un ensemble de test (dans ce cas les données sont artificielles) ; l'autre sur l'évaluation du pouvoir prédictionnel des modèles sur les données artificielles ou réelles. Nous donnons dans le chapitre suivant les protocoles d'expérimentation, les résultats et leurs interprétations.

10 Protocoles, résultats et interprétations ⋆

Sommaire

Résumé

L'évaluation des modèles appris et leur qualité de prédiction sont au centre de ce chapitre. Nous montrons les résultats obtenus sur diverses expérimentations menées sur des données artificielles et réelles. Ces résultats confortent le bon comportement de notre méthode de reconstruction et notre choix d'utiliser l'inférence grammaticale stochastique. La dernière expérimentation indique, par les taux de succès obtenus en prédiction que notre méthode a de meilleures performances que celles recensées dans la littérature.

10.1 Introduction ⋆

Alors que le chapitre précédent nous a permis de décrire les données que nous utilisons dans les expérimentations, ce chapitre va nous permettre de décrire les expérimentations elles-mêmes. Nous faisons volontairement la distinction entre données réelles et données artificielles pour des raisons de protocoles d'évaluation différents. Nous donnons ci-après les protocoles, résultats et interprétations des expérimentations sur données artificielles puis sur données réelles. Il existe dans notre travail trois niveaux de validation de la reconstruction des *logs* : un premier niveau vu dans la section 3.3 qui compare la structure des visites détectées avec et sans méthode de reconstruction ; un deuxième qui traite de l'écart d'un modèle appris par rapport à une distribution cible (estimée par un ensemble de test) ; enfin un dernier niveau qui utilise le modèle appris pour prédire la prochaine page dans la navigation de l'utilisateur.

Le problème de l'évaluation se pose néanmoins pour les deuxième et troisième niveaux : en effet, il n'existe pas d'ensemble de test cohérent aux modèles appris. En

effet, l'ensemble d'apprentissage passé à l'algorithme porte soit sur les *logs* bruts (ce sont les enregistrements du serveur filtrés), soit reconstruits par notre méthode. Dans le dernier cas, nous pensons que l'automate appris modélise au mieux le comportement des utilisateurs, par contre, dans l'autre cas, il est clair que la modélisation représente l'activité du serveur, et donc une activité induite par un ensemble *utilisateur-machines intermédiaires-caches*. Dans ce contexte, il est donc difficile de comparer les modèles sur les données brutes ou reconstruites, nous le faisons néanmoins en introduisant un biais en testant les modèles sur des ensembles de tests bruts ou reconstruits par notre méthode. Nous ne pouvons donc que tester ceci sur des données artificielles pour lesquelles nous pouvons dégrader volontairement les séquences pour les reconstruire ensuite. Plus la reconstruction se comporte comme nous l'espérons, moins le biais sera important. Les résultats obtenus sont intéressants. Enfin, nous décidons de ne plus évaluer le modèle en tant qu'objet mais plutôt de l'utiliser pour détecter la prochaine page que va demander l'utilisateur.

Nous présentons tout d'abord une expérimentation qui évalue les modèles sur des données artificielles puis des expérimentations où l'on prédit la page suivante : tout d'abord sur des données artificielles, puis sur des données réelles.

10.2 Évaluation en tant que modèle appris ⋆

Le biais introduit lors de l'évaluation du modèle est important ; pour garder un contrôle permanent nous décidons de simplifier les choses en classant (coloriant) les pages en deux catégories pour chacun des sites Eurise et Agora21 : les pages contenant le mot *apprentissage* sur le premier site seront noires, les autres bleues ; les pages contenant l'expression *développement durable* sur le deuxième site seront noires, les autres bleues. En faisant de la sorte, nous obtenons pour chacun des sites, deux ensembles de pages sensiblement de même taille (pour ne pas introduire de biais supplémentaire). Les visites précédemment générées (cf. section 3.3) sont coloriées *via* cette technique et scindées en ensembles d'apprentissage et de test. Nous apprenons un modèle avec l'algorithme Alergia pour les visites brutes et les visites reconstruites et nous calculons la perplexité (et la distance d_2 dans le cas[1] des données Eurise) entre les modèles appris et les distributions cibles estimées par le test. Les résultats sont donnés dans les tables 10.1. L'abréviation réf. (respectivement dég. et rec.) sont mises pour les *logs* de référence (resp. dégradés et reconstruits). Pour ne pas souffrir de variations contenues dans les données d'expérimentations, nous utilisons une validation croisée, c'est à dire que nous partageons les ensembles de départ en parties de tailles égales et nous menons plusieurs expérimentations pour que chaque partie soit l'ensemble de test (le reste étant l'ensemble d'apprentissage).

Nous pouvons remarquer que les résultats obtenus valident notre reconstruction. En effet, les résultats mis en avant sur les *logs* reconstruits sont meilleurs que ceux sur les données dégradées que l'on devrait avoir dans la vie réelle. De plus, dans le cas des

[1] Cette expérimentation menée en début de thèse ne donnait pas les résultats de distance. Pour des questions de coût calculatoire, nous avons décidé de calculer la distance uniquement sur ces données.

Logs	réf.	dég.	rec.
EURISE	2,31	3,39	2,56
AGORA21	2,08	2,57	2,31

(a) Perplexité moyenne

Logs	réf.	dég.	rec.
EURISE	1,07	1,43	1,41

(b) Distance moyenne

TAB. 10.1 – *Perplexité et distance moyennes entre modèles appris et distributions cibles*

données EURISE, la distance montre un écart plus important entre les *logs* de référence et ceux de notre méthode, et un gain plus faible par rapport à ceux de la vie réelle. Ces écarts seront consolidés dans les expériences de prédiction où le taux de succès montrera un gain significatif mais encore loin de la perfection.

10.3 Utilisation des modèles appris en prédiction ⋆

Dans le contexte qui nous intéresse maintenant, nous utilisons les modèles appris en prédiction : en fonction d'un certain nombre de pages d'une visite en cours, nous donnons la page la plus probable comme étant la suite de la navigation de l'utilisateur. Le protocole des expérimentations suivantes est simple : en même temps que les séquences de pages représentant les visites sont analysées par l'automate, nous vérifions que la page « à venir » est la plus probable dans l'automate. Nous calculons ensuite un taux de succès en comptant comme correct la bonne prévision de l'automate pour la page donnée. Cette tâche est difficile et nous le verrons dans la dernière expérimentation, menée sur ce protocole, où les résultats de la littérature sont assez faibles.

Dans un premier temps, nous travaillons sur les mêmes données que l'expérimentation précédente, à savoir les données générées sur le site EURISE. Les résultats sont donnés dans la figure 10.1. Nous remarquons que les résultats obtenus en taux de succès de prédiction de la page suivante sont meilleurs avec notre méthode de reconstruction des données. Nous remarquons aussi que les modèles appris ont une meilleure qualité de prédiction en début de visite qu'après quelques pages. Ce problème est intrinsèque à la tâche : dans [GH03], les auteurs mentionnent ne pas pouvoir donner d'estimation correcte pour les pages placées après le rang 7 avec leur méthode.

Dans un deuxième temps, nous réalisons le même type d'expérimentation, cette fois-ci sur les données réelles du même site. Nous utilisons la structure du site et les *logs* présentés dans le chapitre précédent. Les résultats sont donnés dans la figure 10.2. En comparant les résultats des figures 10.1 et 10.2, nous pouvons affirmer que notre méthode de reconstruction est intéressante pour la tâche sur laquelle nous travaillons.

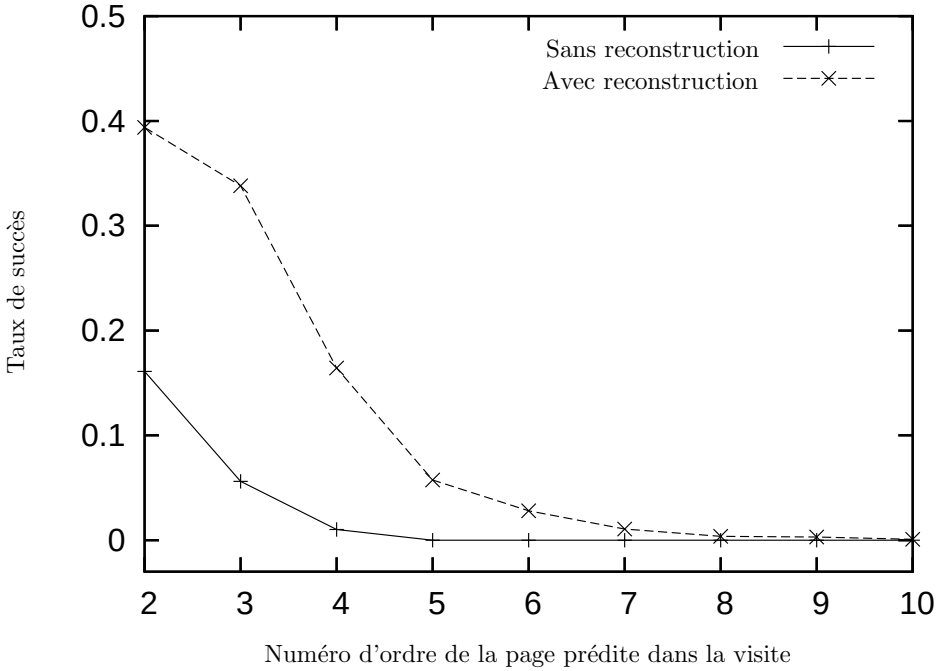


Fig. 10.1 – *Taux de succès en prédiction sur les données artificielles*

De plus, sur les données réelles, les aléas des données donnent des taux de succès répartis de manière saccadée ; sur les données artificielles, la courbe est plus lisse.

La dernière expérimentation que nous avons menée sur ce type de données prend en considération les résultats obtenus par les auteurs de [GH03]. Dans cet article, Géry *et al.*[2] décrivent les données du site MRIM — et d'un site (VTT) dont nous n'avons pas les données — et les méthodes utilisées pour prédire la page suivante dans la navigation de l'utilisateur. Ils utilisent *les règles d'association*, la recherche de *séquence fréquente* et une généralisation qu'ils proposent, les *séquences généralisées fréquentes*[3]. Après nombre d'expérimentations, en faisant varier certains paramètres, ils nous livrent leurs meilleurs résultats en terme de taux de succès. La table 10.2 nous donne les taux de succès de leur méthode comparée à la nôtre. Les résultats sont

	Géry *et al.*	notre méthode
Taux de succès (%)	21,56	28,92

Tab. 10.2 – *Taux de succès en prédiction sur les données* MRIM

meilleurs avec notre méthode que celles mises en place dans l'article qui nous sert de

2 Nous tenons à remercier grandement Mathias Géry pour nous avoir mis à disposition, non seulement les *logs* du site MRIM mais aussi le site lui-même.

3 Nous invitons le lecteur intéressé à lire l'article pour plus de détails

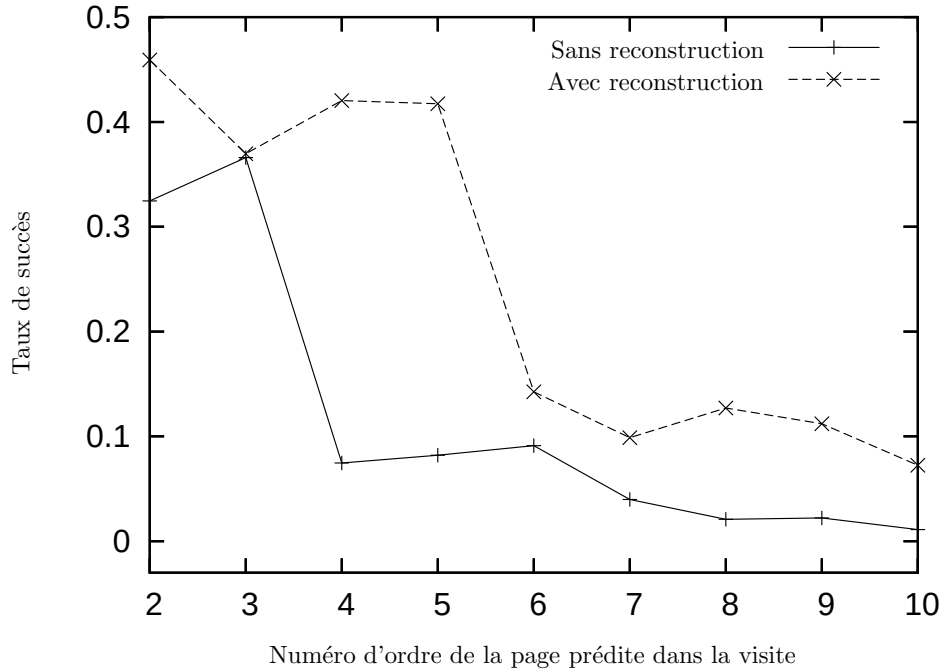


FIG. 10.2 – *Taux de succès en prédiction sur les données réelles*

référence. La reconstruction des logs que nous faisons permet de garder une certaine qualité d'information dans les visites utilisateur que nous traitons, qualité qui est perdue dans des visites classiques. De plus, le choix de l'inférence grammaticale stochastique, justifié par la recherche de modèles intelligibles et l'absence de données négatives est conforté dans ces résultats : les modèles structurels sont utilisables sur des masses de données importantes telles que celles issues du *Web*.

10.4 Conclusion ⋆

Dans ce chapitre, nous avons donné une série de résultats d'expérimentations en ayant au préalable décrit les protocoles utilisés. De plus, dans la dernière expérimentation, le protocole est le même que celui des auteurs auxquels nous faisons référence. Les résultats que nous obtenons après la phase d'apprentissage confortent ceux donnés sur la morphologie des visites reconstruites. Ces résultats indiquent que le processus entier d'ECD que nous utilisons est bon : du pré-traitement des données (avec la reconstruction) au choix de l'inférence grammaticale qui est donc appropriée à ces données de taille conséquente.

Conclusion ⋆

1 Synthèse ⋆

Le pré-traitement des données du *Web*, en vue de l'utilisation de celles-ci dans un processus d'extraction de connaissance à partir de données (quelques soient les méthodes employées pour cela), est un vaste domaine. Dans ce mémoire, nous avons étudié les différents types de données disponibles, pour finalement nous concentrer sur les enregistrements des transactions du serveur. Ce choix, que nous avons motivé tout au long de notre travail, est inhérent à la problématique proposée à savoir développer des méthodes en vue de l'adaptation d'un site web quelconque.

Dans notre étude, lors de la phase d'apprentissage, nous avons choisi d'utiliser l'inférence grammaticale stochastique puisqu'elle est souvent mise en avant comme réponse à un problème d'apprentissage automatique avec un contexte dans lequel il n'existe pas de données négatives. Dans notre cas, nous avons aussi choisi d'utiliser des techniques d'inférence stochastique pour que les modèles appris soient intelligibles. En effet, la structure des automates appris permet une visualisation simple du modèle, ce qui rend l'étape finale du processus d'ECD (*validation*) plus facile.

Dans cette partie nous avons présenté l'état de l'art en apprentissage automatique en donnant les définitions et aspects théoriques classiques. Une des contributions de notre travail a été de mettre en place le calcul de distances entre modèles de langages calculées sur les automates. Des travaux récents [CMR06] reprennent les méthodes décrites dans [LPN99], dont nous nous sommes servi pour calculer les distances d_2 et d_{2p}, pour calculer les distances d_{2k} pour tout k appartenant à $\mathbb{N}^*$. Ceci montre l'engouement, encore d'actualité, autour de ces problématique d'évaluation de l'écart entre modèles. En plus des résultats théoriques concernant les distances, nous avons expérimenté ces mesures dans diverses tâches pour valider leur comportement. Nous avons montré notamment qu'en apprentissage automatique où la d_{KL} (ou la perplexité) est une référence, il est plus judicieux de se servir des distances qui peuvent, elles, évaluer un modèle non lissé.

La troisième partie, quant à elle, nous a donné les résultats d'expérimentations complexes mettant en œuvre les méthodes développées dans les deux parties précédentes. Nous avons décrit de manière approfondies les données que nous avons utilisées, aussi bien celles générées par nos soins, que celles issues d'un vrai site web, ou celles mises à notre disposition par les auteurs de [GH03]. Nous avons montré que notre contribution pour la reconstruction des logs est adéquate de plusieurs façon. Dans la première partie,

nous avons comparé la morphologie des visites détectées, reconstruites et de référence. Dans la troisième partie, nous avons montré que les modèles appris depuis les données reconstruites étaient meilleurs (en terme de distance, de perplexité) que les modèles sur les données de base. Enfin dans la dernière expérimentation nous avons évalué le pouvoir de prédiction de ces modèles et montré qu'ils étaient meilleurs que ceux de la littérature.

Nous pensons avoir apporté un nouveau point de vue pour le prétraitement de données web et la reconstruction des logs, ainsi qu'une contribution importante pour l'évaluation des modèles appris grâce à la distance d'automates. La tendance actuelle, grâce aux puissances de calcul existantes, est de traiter de grands ensembles de données et d'obtenir des modèles de plus en plus intelligibles. Nous montrons aussi que l'inférence grammaticale est utilisable même sur des volumes de données importants : ce qui est assez nouveau.

Ce mémoire de thèse n'a bien sûr pas pour but de résoudre de manière définitive le vaste problème de l'adaptation automatique des sites Web. Il reste encore de nombreux travaux à mener et nous proposons maintenant quelques pistes de recherche ouvertes par ces travaux ou bien qui pourraient les compléter.

2 Perspectives ⋆

À l'heure actuelle, de plus en plus de sites web utilisent des pages web dynamiques côté serveur pour adapter le contenu des pages à l'utilisateur. Ces pages, comme nous l'expliquons dans le chapitre 3 ne peuvent pas être mises en cache, car leur contenu est différent suivant la personne qui émet la requête. Les navigateurs riches, c'est à dire qui acceptent des requêtes `HTTP` asynchrones pour obtenir le plus souvent des données variables sous forme XML, se développent à l'encontre du schéma précédent : une page contient souvent une partie qui change d'un utilisateur à l'autre, mais surtout la quasi-totalité qui reste inchangée. Ainsi les données XML peuvent pour une grande partie être mises en cache. Une idée de travail intéressant serait de pouvoir étudier à grande échelle tous ces problèmes de mise en cache suivant les protocoles employés et la richesse des clients utilisés. Des travaux mentionnent des caches hiérarchiques [CDN+96], ou encore le recours à une architecture partagée [SM06].

Dans [Tho00], le critère de fusion d'un couple d'états présenté pour l'algorithme MDI utilise un calcul de divergence de Kullback-Leibler. L'utilité de ce test, en comparaison à celui de l'algorithme ALERGIA est de prendre en considération la totalité de la distribution de probabilité. Néanmoins, lors d'une opération de fusion et déterminisation pour un couple d'états, une partie limitée de l'automate change. En relevant ce fait, il est possible de calculer efficacement l'accroissement de divergence entre un automate et un de ses automates quotients. Dans nos travaux, en mettant en avant une distance ne souffrant pas des problèmes de la d_{KL}[1] qui n'est pas une vraie distance,

[1]Nous rappelons au lecteur que dans le cas d'un calcul de d_{KL} entre un automate et un de ses

une amélioration serait d'évaluer un algorithme ayant pour critère de fusion, celui de MDI dans lequel la d_{KL} est remplacée par la d_2. Cet algorithme a été implanté par nos soins, mais à l'heure actuelle, comme aucun calcul incrémental de la d_2 n'a été mis au point, la complexité calculatoire ne nous permet pas de l'utiliser sur de vraies données. Nous tenons à mentionner que sur des petits objets (la grammaire de REBER[2]), les résultats sont aussi bons qu'avec la d_{KL}. Un point intéressant de recherche serait donc de calculer incrémentalement la d_2. À partir de là, la d_2 étant une vraie distance, il serait sûrement possible d'accélérer encore MDI en prenant en considération l'inégalité triangulaire, qui nous autoriserait à ne pas considérer certains couples d'états comme candidats.

quotients, il est montré que la grandeur est toujours finie.

[2]cf. chapitre 8

Annexes

1 Preuve : liens entre fonctions de probabilité et de probabilité préfixielle

Pour certaines des preuves suivantes nous avons besoin d'écrire la probabilité d'un mot sous sa forme structurelle, à savoir un produit de probabilité de transition d'un automate. Nous redonnons ci-après la propiété de réécriture itérative.

$$\forall w \in \Sigma^*, \mathrm{p}(q,w) = \prod_{i=1}^{|w|} \left(\mathrm{p}^{\bullet}\left(\delta\big(q, \underset{i-1}{\mathrm{Pr}}(w)\big), w[i]\right)\right) \times \mathrm{f}\big(\delta(q,w)\big)$$

Preuve : Cette propriété est simplement la réécriture itérative de la définition récursive du calcul de la probabilité d'un mot dans un automate. □

Nous donnons de nouveau les divers liens entre fonction de probabilité p et fonction de probabilité préfixielle π ainsi que la preuve de chacun.

1. $\forall w \in \Sigma^*, \pi(w) = \mathrm{p}(w) + \sum_{a \in \Sigma} \pi(wa)$

   Preuve : Par définition,

   $$\forall w \in \Sigma^*, \pi(w) = \sum_{x \in \Sigma^*} \mathrm{p}(wx)$$

   En sortant λ de l'ensemble Σ^*, on obtient :

   $$\forall w \in \Sigma^*, \pi(w) = \mathrm{p}(w) + \sum_{\substack{x \in \Sigma^* : \\ x \neq \lambda}} \mathrm{p}(wx)$$

   De plus,

   $$\forall x \in \Sigma^*, x \neq \lambda,\ \exists a \in \Sigma, z \in \Sigma^* |\, x = az$$

   Ce qui donne,

   $$\forall w \in \Sigma^*, \pi(w) = \mathrm{p}(w) + \sum_{a \in \Sigma} \sum_{z \in \Sigma^*} \mathrm{p}(waz)$$

En appliquant la définition de la fonction préfixielle, on obtient le résultat cherché :

$$\forall w \in \Sigma^*, \pi(w) = \mathrm{p}(w) + \sum_{a \in \Sigma} \pi(wa)$$

□

2. $\forall w \in \Sigma^*, \mathrm{p}(w) = \pi(w) \times \mathrm{f}(\delta(q_0, w))$

Preuve : Développons le deuxième terme,

$$\begin{aligned}\forall w \in \Sigma^*, \pi(w) \times \mathrm{f}\big(\delta(q_0, w)\big) &= \sum_{x \in \Sigma^*} \big(\mathrm{p}(wx)\big) \times \mathrm{f}\big(\delta(q_0, w)\big) \\ &= \sum_{x \in \Sigma^*} \Big(\mathrm{p}(wx) \times \mathrm{f}\big(\delta(q_0, w)\big)\Big)\end{aligned}$$

En appliquant la réécriture itérative, la valeur devient :

$$\sum_{x \in \Sigma^*} \left(\prod_{i=1}^{|w|} \Big(\mathrm{p}^\bullet\Big(\delta\big(q_0, \underset{i-1}{\mathrm{Pr}}(w)\big), w[i]\Big)\Big) \prod_{i=1}^{|x|} \Big(\mathrm{p}^\bullet\Big(\delta\big(q_0, w \underset{i-1}{\mathrm{Pr}}(x)\big), x[i]\Big)\Big) \times \mathrm{f}\big(\delta(q_0, wx)\big) \times \mathrm{f}\big(\delta(q_0, w)\big)\right)$$

En changeant l'ordre des termes,

$$\prod_{i=1}^{|w|} \Big(\mathrm{p}^\bullet\Big(\delta\big(q_0, \underset{i-1}{\mathrm{Pr}}(w)\big), w[i]\Big)\Big) \times \mathrm{f}\big(\delta(q_0, w)\big) \times \sum_{x \in \Sigma^*} \left(\prod_{i=1}^{|x|} \Big(\mathrm{p}^\bullet\Big(\delta\big(q_0, w \underset{i-1}{\mathrm{Pr}}(x)\big), x[i]\Big)\Big) \times \mathrm{f}\big(\delta(q_0, wx)\big)\right)$$

Et donc,

$$\mathrm{p}(w) \times \sum_{x \in \Sigma^*} \Big(\mathrm{p}\Big(\delta\big(q_0, w \underset{i-1}{\mathrm{Pr}}(x)\big), x\Big)\Big)$$

Un automate définissant une distribution de probabilité est tel que chaque état définit aussi une telle distribution, la dernière somme est donc égale à 1, on obtient ainsi :

$$\forall w \in \Sigma^*, \pi(w) \times \mathrm{f}\big(\delta(q_0, w)\big) = \mathrm{p}(w)$$

□

3. $\forall w \in \Sigma^*, \forall a \in \Sigma, \pi(wa) = \pi(w) \times \mathrm{p}^\bullet(\delta(q_0, w), a)$

Preuve : Par définition,

$$\forall w \in \Sigma^*, \forall a \in \Sigma, \pi(wa) = \sum_{x \in \Sigma^*} p(wax)$$

En appliquant la réécriture itérative, la valeur devient :

$$\sum_{x\in\Sigma^*}\prod_{i=1}^{|w|}\Big(\mathrm{p}^\bullet\big(\delta(q_0,\Pr_{i-1}(w)),w[i]\big)\Big)\times\mathrm{p}^\bullet\big(\delta(q_0,w),a\big)\times\prod_{i=1}^{|x|}\Big(\mathrm{p}^\bullet\big(\delta(q_0,wa\Pr_{i-1}(x)),x[i]\big)\Big)\times\mathrm{f}\big(\delta(q_0,wax)\big)$$

En réorganisant les termes,

$$\pi(w)\times\mathrm{p}^\bullet\big(\delta(q_0,w),a\big)\times\sum_{x\in\Sigma^*}\prod_{i=1}^{|x|}\Big(\mathrm{p}^\bullet\big(\delta(q_0,wa\Pr_{i-1}(x)),x[i]\big)\Big)\times\mathrm{f}\big(\delta(q_0,wax)\big)$$

En définissant $q_{wa}=\delta(q_0,wa)$, on obtient :

$$\pi(w)\times\mathrm{p}^\bullet\big(\delta(q_0,w),a\big)\times\sum_{x\in\Sigma^*}\prod_{i=1}^{|x|}\Big(\mathrm{p}^\bullet\big(\delta(q_{wa},\Pr_{i-1}(x)),x[i]\big)\Big)\times\mathrm{f}\big(\delta(q_{wa},x)\big)$$

Enfin,

$$\pi(w)\times\mathrm{p}^\bullet\big(\delta(q_0,w),a\big)\times\sum_{x\in\Sigma^*}\mathrm{p}(q_{wa},x)$$

Le dernier terme valant 1 (distribution de probabilité depuis l'état q_{wa}), l'égalité est prouvée. □

2 Preuve : d_{2p} définie sur les distributions de probabilité est finie, est une vraie distance, et est calculable sur les DPFA

Nous allons prouver ici les trois points de l'énoncé du théorème 8.2. Nous redonnons auparavant l'écriture de la d_{2p} et son développement en faisant apparaître la co-émission préfixielle.

$$\begin{aligned}\mathrm{d}_{2p}(\mathcal{A},\mathcal{B}) &= \sqrt{\sum_{w\in\Sigma^*}\big(\pi_\mathcal{A}(w)-\pi_\mathcal{B}(w)\big)^2}\\ &= \sqrt{\mathrm{CoEmPref}(\mathcal{A},\mathcal{A})+\mathrm{CoEmPref}(\mathcal{B},\mathcal{B})-2.\,\mathrm{CoEmPref}(\mathcal{A},\mathcal{B})}\end{aligned}$$

Nous montrons ci-après les diverses propriétés du théorème.

1. Montrons que cette grandeur est toujours finie. Pour cela nous montrons facilement que la fonction de co-émission préfixielle est finie.

   Par définition :

   $$\mathrm{CoEmPref}(\mathcal{A},\mathcal{B})=\sum_{q\in Q_\mathcal{A}}\sum_{q'\in Q_\mathcal{B}}\big(\eta_{qq'}\big)$$

Les coefficients η étant finis, la mesure d_{2p} est donc finie.

2. Reprenons point par point les propriétés pour la distance d_{2p} sur les DPFA. Nous considérons comme éléments $x = \mathcal{A}$, $y = \mathcal{B}$, $z = \mathcal{C}$, des distributions de probabilités représentées par des DPFA.

   (a) $\mathrm{d}_{2p}(\mathcal{A}, \mathcal{B}) \geq 0$: La somme de carrés dans $\mathbb{R}$ est trivialement positive ou nulle.

   (b) $\mathrm{d}_{2p}(\mathcal{A}, \mathcal{B}) = 0 \iff \mathcal{A} = \mathcal{B}$: La somme de carrés dans $\mathbb{R}$ est nulle si et seulement si tous les termes $\pi_{\mathcal{A}}(w) - \pi_{\mathcal{B}}(w)$ sont nuls, c'est à dire si et seulement si $\mathrm{p}_{\mathcal{A}} = \mathrm{p}_{\mathcal{B}}$. Les distributions $\mathcal{A}$ et $\mathcal{B}$ sont donc égales.

   (c) $\mathrm{d}_{2p}(\mathcal{A}, \mathcal{B}) = \mathrm{d}_{2p}(\mathcal{B}, \mathcal{A})$: trivial.

   (d) $\mathrm{d}_{2p}(\mathcal{A}, \mathcal{B}) \leq \mathrm{d}_{2p}(\mathcal{A}, \mathcal{C}) + \mathrm{d}(\mathcal{C}, \mathcal{B})$:
   La preuve donnée dans le cas de la distance euclidienne peut être reprise dans ce cas, en changeant simplement les références à la probabilité des mots par la fonction préfixielle.

3. De la même manière, la calculabilité de la distance tient dans la résolution du système linéaire des coefficients η.

Bibliographie

[ABCF94] G. Antoniol, F. Brugnara, M. Cettolo, et M. Federico, *Language Model Estimations and Representations for Real-time Continuous Speech Recognition*, ICSLP (Yokohama), 1994, p. 859–862.

[Ago02] Agora21, *Agora21 web site*, http://www.agora21.org/, 2002.

[Aka] Akamai, *Akamai technologies, web site*, http://www.akamai.com.

[And90] J. André, *Petites leçons de typographie*, Irisa-Hebdo, revisé en 2003 et 2006, 1990.

[AS94] R. Agrawal et R. Srikant, *Fast Algorithms for Mining Association Rules in Large Databases.*, Proceedings of the 20th International Conference on Very Large Data Bases, VLDB'94 (Santiago de Chile, Chile) (J. B. Bocca, Matthias Jarke, et Carlo Zaniolo, eds.), M. Kaufmann, sep 1994, p. 487–499.

[ASA+95] M. Abrams, C. R. Standrige, G. Abdulla, S. Williams, et E. A. Fox, *Cache Proxies: Limitations and Potentials*, Proceedings of the fourth Interational World-Wide Web Conference (Boston, MA), 1995.

[Bar] B. L. Barrett, *The webalizer, web site*, http://www.mrunix.net/webalizer/.

[BL99] J. Borges et M. Levene, *Data Mining of User Navigation Patterns*, WEBKDD, 1999, p. 92–111.

[BLP+03] S. Bidel, L. Lemoine, F. Piat, T. Artières, et P. Gallinari, *Statistical Machine Learning for Tracking Hypermedia User Behaviour*, MLIRUM - UM Workshop (Pittsburgh), Juin 2003.

[Bou] inc. Bouttell.com, *Wusage, web site*, http://www.boutell.com/wusage/.

[CBC03] K. Chevalier, C. Bothorel, et V. Corruble, *Discovering Rich Navigation Pattern on a Web Site*, Discovery Science, 6th International Conference, DS 2003, Sapporo, Japan, October 17-19,2003, Proceedings (Gunter Grieser, Yuzuru Tanaka, et Akihiro Yamamoto, eds.), Lecture Notes in Computer Science, vol. 2843, Springer, 2003.

[CDN+96] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, et K. J. Worrell, *A Hierarchical Internet Object Cache*, USENIX Annual Technical Conference, 1996, p. 153–164.

[CG98] S. F. Chen et J. Goodman, *An Empirical Study of Smoothing Techniques for Language Modeling*, Technical Report TR-10-98, Computer Science Group, Harvard University, 1998.

[CG99] ———, *An Empirical Study of Smoothing Techniques for Language Modeling*, Computer Speech & Language **13** (1999), no. 4, 359–393.

[Cho57] N. Chomsky, *Syntactic Structures*, Mouton, La Haye, 1957.

[Cir] Ciridd, *Centre International de Ressources et d'Innovation pour le Développement Durable, web site*, http://www.ciridd.org.

[CMR06] C. Cortes, M. Mohri, et A. Rastogi, *On the Computation of Some Standard Distances Between Probabilistic Automata*, Implementation and Application of Automata, 11th International Conference, CIAA 2006, Proceedings (Taipei, Taiwan) (O. H. Ibarra et H.-C. Yen, eds.), vol. 4094, Springer-Verlag, aug 2006, ISBN : 3-540-37213-X.

[CMS99] R. Cooley, B. Mobasher, et J. Srivastava, *Data Preparation for Mining World Wide Web Browsing Patterns*, Knowledge and Information Systems **1** (1999), no. 1, 5–32.

[CO94a] R. Carrasco et J. Oncina, *Learning Stochastic Regular Grammars by Means of a State Merging Method*, in Carrasco and Oncina [CO94b], p. 139–150.

[CO94b] R. C. Carrasco et J. Oncina (eds.), *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, Proceedings*, Lecture Notes in Artificial Intelligence, no. 862, Berlin, Heidelberg, Springer-Verlag, Septembre 1994.

[CO99] R. Carrasco et J. Oncina, *Learning Deterministic Regular Grammars From Stochastic Samples in Polynomial Time*, RAIRO, ITA **33** (1999), no. 1, 1–19.

[CP95] L. D. Catledge et J. E. Pitkow, *Characterizing Browsing Strategies in the World-Wide Web*, Computer Networks and ISDN Systems **27** (1995), no. 6, 1065–1073.

[CPY96] M.-S. Chen, J. S. Park, et P. S. Yu, *Data Mining for Path Traversal Patterns in a Web Environment*, Proceedings of the 16th IEEE International Conference on Distributed Computing Systems, 1996, p. 385–392.

[CRJ02] R. C. Carrasco et J. R. Rico-Juan, *A Similarity Between Probabilistic Tree Languages: Application to XML Document Families*, Pattern Recognition, in Press **36** (2002), no. 9, 2197–2199.

[CSM97] R. Cooley, J. Srivastava, et B. Mobasher, *WEB Mining: Information and Pattern Discovery on the World Wide Web*, Proceedings of the 9th International Conference on Tools With Artificial IntelligenceICTAI'97, 1997.

[CT04] A. Clark et F. Thollard, *PAC-learnability of Probabilistic Deterministic Finite State Automata*, Journal of Machine Learning Research (2004), no. 5, 473–497.

[DDE05] P. Dupont, F. Denis, et Y. Esposito, *Links Between Probabilistic Automata and Hidden Markov Models: Probability Distributions, Learning Models and Induction Algorithms*, Pattern Recognition: Special Issue on Grammatical Inference Techniques & Applications **38** (2005), no. 9, 1349–1371.

[DM98] P. Dupont et L. Miclet, *Inférence Grammaticale Régulière : Fondements Théoriques Et Principaux Algorithmes*, Tech. Report RR-3449, INRIA, 1998.

[Doy91] A. Conan Doyle, *A scandal in bohemia*, in The Adventures of Sherlock Holmes, 1891.

[Eur02] Eurise, *Web site*, http://eurise.univ-st-etienne.fr, 2002.

[FMK02] E. Frias-Martinez et V. Karamcheti, *A Prediction Model for User Access Sequences*, WEBKDD Workshop: Web Mining for Usage Patterns and User Profiles, 2002.

[FPSM91] W. Frawley, G. Piatetsky-Shapiro, et C. Matheus, *Knowledge Discovery in Databases*, ch. Knowledge Discovery in Databases: An Overview, p. 1–27, AAAI/MIT Press, 1991.

[Gar05] J. J. Garrett, *Ajax: A new approach to web applications*, http://www.adaptivepath.com/publications/essays/archives/-000385print.php, feb 2005.

[GE03] A. Gagneux et H. Emptoz, *Web Site: a Strcutured Document*, th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK, 2003, p. 1158–1162.

[GH03] M. Géry et H. Haddad, *Evaluation of Web Usage Mining Approaches for User's Next Request Prediction*, Proceedings of the Fifth International Workshop on Web Information and Data ManagementWIDM'03 (New Orleans), 2003, p. 74–81.

[Gol78] E. M. Gold, *Complexity of Automaton Identification From Given Data*, Information and Control **37** (1978), 302–320.

[Goo] Google, *Web site*, http://www.google.com.

[GSL+02] J. H. Goldberg, M. J. Stimson, M. Lewenstein, N. Scott, et A. M. Wichansky, *Eye Tracking in Web Search Tasks: Design Implications*, ETRA 02: Proceedings of the Symposium on Eye Tracking Research & Applications (New York, NY, USA), ACM Press, 2002, p. 51–58.

[HBS03] A. Habrard, M. Bernard, et M. Sebban, *Improvement of the State Merging Rule on Noisy Data in Probabilistic Grammatical Inference*, Machine Learning: ECML 2003, 14th European Conference on Machine Learning, Proceedings (Cavtat-Dubrovnik, Croatia) (N. Lavrac, D. Gamberger, L. Todorovski, et H. Blockeel, eds.), Lecture Notes in Computer Science, vol. 2837, Springer-Verlag, september 2003, p. 169–180.

[HBS05] ———, *Detecting Irrelevant Subtrees to Improve Probabilistic Learning From Tree-structured Data*, Fundamenta Informaticae **66** (2005), no. 1-2, 103–130.

[Hir92] L. Hirschman, *Multi-Site Data Collection for a Spoken Language Corpus*, Proceedings of DARPA Speech and Natural Language Workshop, 1992, p. 7–14.

[HO04] C. de la Higuera et J. Oncina, *Learning Stochastic Finite Automata.*, Grammatical Inference: Algorithms and Applications, 7th International Colloquium, ICGI 2004, Athens, Greece. Proceedings (Berlin, Heidelberg) (G. Paliouras et Y. Sakakibara, eds.), Lecture Notes in Artificial Intelligence, no. 3264, Springer-Verlag, Octobre 2004, p. 175–186.

[Hoe63] W. Hoeffding, *Probability Inequalities for Sums of Bounded Random Variables*, American Statistical Association Journal (1963), 13–30.

[HOV96] C. de la Higuera, J. Oncina, et E. Vidal, *Identification of dfa: data-dependent vs data-independent algorithms*, Grammatical Inference: Learning Syntax from Sentences, 3rd International Colloquium, ICGI-96, Montpellier, France, Proceedings (Berlin, Heidelberg) (L. Miclet et C. de la Higuera, eds.), Lecture Notes in Artificial Intelligence, no. 1147, Springer-Verlag, Septembre 1996, p. 313–325.

[HS66] J. Hartmanis et R. E. Stearns, *Algebraic Struture Theory of Sequential Machines*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1966.

[HT00] C. de la Higuera et F. Thollard, *Identication in the Limit With Probability One of Stochastic Deterministic Finite Automata*, Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000, Lisbon, Portugal, Proceedings (Berlin, Heidelberg) (A. L. Oliveira, ed.), Lecture Notes in Artificial Intelligence, no. 1891, Springer-Verlag, Septembre 2000, p. 15–24.

[HTV+05] C. de la Higuera, F. Thollard, E. Vidal, F. Casacuberta, et R. Carrasco, *Probabilistic Finite-state Machines - Part I*, IEEE Transactions on Pattern Analysis and Machine Intelligence (2005), 1013–1025.

[IET95] RUN Group Of IETF, *Netiquette guidelines*, http://www.ietf.org/rfc/-rfc1855.txt, oct 1995.

[Int99] Ecma International, *Ecmascript language specification 3rd edition*, http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf, 1999.

[ISO86] ISO, *Information processing – text and office systems – standard generalized markup language (sgml)*, 1986.

[Kat87] S. Katz, *Estimation of Probabilities From Sparse Data for the Language Model Component of a Speech Recognizer*, Transactions on Acoustics, Speech, and Signal Processing, IEEE **35** (1987), no. 3, 400–401.

[KB00] R. Kosala et H. Blockeel, *Web Mining Research: a Survey*, SIGKDD Explorations Newsletter **2** (2000), no. 1, 1–15.

[KD02] C. Kermorvant et P. Dupont, *Stochastic Grammatical Inference With Multinomial Tests*, Grammatical Inference: Algorithms and Applications, 6th International Colloquium: ICGI 2002, Amsterdam, The Netherlands, Proceedings (P. W. Adriaans, H. Fernau, et M. v. Zaanen, eds.), Lecture Notes in Computer Science, no. 2484, Springer-Verlag, Septembre 2002, p. 149–160.

[KMR+94] M. J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, et L. Sellie, *On the Learnability of Discrete Distributions*, Proc. of the 25th Annual ACM Symposium on Theory of Computing, 1994, p. 273–282.

[KPPS94] N. Karampatziakis, G. Paliouras, D. Pierrakos, et P. Stamatopoulos, *Navigation Pattern Discovery Using Grammatical Inference*, in Carrasco and Oncina [CO94b], p. 17–56.

[Kul59] S. Kullback, *Information Theory and Statistics*, John Wiley and Sons., New York, 1959.

[KV89] M. Kearns et L. Valiant, *Cryptographic Limitations on Learning Boolean Formulae and Finite Automata*, 21st ACM Symposium on Theory of Computing, 1989, p. 433–444.

[Lab03] D. Labbe, *Corneille dans l'ombre de Molière*, Histoire d'une recherche, Les Impressions Nouvelles, Bruxelles, 2003.

[Lan92] K. J. Lang, *Random DFA's Can Be Approximately Learned From Sparse Uniform Examples*, 5th ACM Workshop on Computational Learning Theory, 1992, p. 45–52.

[Lev66] V. I. Levenshtein, *Binary Codes Capable of Correcting Deletions, Insertions, and Reversals*, Soviet Physics - Doklady **10** (1966), no. 8, 707–710, Translated from Doklady Akademii Nauk SSSR, Vol. 163 No. 4 pp. 845-848, August 1965.

[LPN99] R.B. Lyngsø, C. N. S. Pedersen, et H. Nielsen, *Metrics and Similarity Measures for Hidden Markov Models*, 7th International Conference on Intelligent Systems for Molecular Biology, ISMB '99 Proceedings (Heidelberg, Germany) (T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H. W. Mewes, et R. Zimmer, eds.), AAAI Press, Menlo Park, CA94025, USA, 1999, p. 178–186.

[LPP98] K. J. Lang, B. A. Pearlmutter, et R. A. Price, *Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm*, Grammatical Inference, Lecture Notes in Artificial Intelligence, no. 1433, Springer-Verlag, 1998, p. 1–12.

[Luo95] A. Luotonen, *The Common Log File Format*, http://www.w3.org/Daemon/User/Config/Logging.html, 1995.

[MBT+04] Meurthe, Berlier, Thibaudeau, Emmery, Réal, Bigot de Preameneu, Régnier, et Abrial, *Code civil des français*, promulgué par Bonaprte, mar 1804.

[MDLN01] B. Mobasher, H. Dai, T. Luo, et M. Nakagawa, *Effective Personalization Based on Association Rule Discovery from Web Usage Data*, Proceedings of the third International Workshop on Web Information and Data Management, WIDM'01 (Atlanta, GA), 2001, p. 9–15.

[MH04a] T. Murgue et C. de la Higuera, *Distances Between Distributions: Comparing Language Models*, Proc. of Syntactical and Structural Pattern Recognition Workshop (Springer-Verlag Berlin Heidelberg 2004, ed.), LNCS, no. 3138, 2004, p. 269–277.

[MH04b] ———, *Distances entre distributions de probabilité : comparaison de modèles de langages stochastiques*, Conférence d'Apprentissage (CAp'04) (Montpellier, France), Presses Universitaires de Grenoble, 2004, p. 97–112.

[MJ05] T. Murgue et P. Jaillon, *Data Preparation and Structural Models for Web Usage Mining*, Sciences of Electronic, Technologies of Information and Telecommunications (SETIT'05) (Sousse, Tunisie), 2005.

[MRI02] MRIM, *Web site*, http://www-clips.imag.fr/mrim/, 2002.

[Mur05a] T. Murgue, *De l'importance du pré-traitement des données pour l'utilisation de l'inférence grammaticale en Web Usage Mining*, Atelier sur la modélisation utilisateurs et personnalisation de l'interaction homme-machine (EGC'05), 2005.

[Mur05b] ———, *Log Pre-Processing and Grammatical Inference for Web Usage Mining*, Workshop on Machine Learning for User Modeling: Challenges – UM 2005, 2005.

[Mur06] ———, *Extraction Des Connaissances : État Et Perspectives*, RNTI, vol. E5, ch. Modélisation d'utilisateurs et Personnalisation de l'Interaction Homme-Machine, Cépaduès, Toulouse, France, 2006, ISBN :2.85428.707.X.

[ONU48] ONU, *Déclaration universelle des droits de l'homme*, résolution 217 A (III), dec 1948, Les 58 états membres de l'ONU.

[PE97] M. Perkowitz et O. Etzioni, *Adaptive Web Sites: an AI Challenge*, Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97 (Nagoya, Japan) (M. Kaufmann, ed.), vol. 1, aug 1997, p. 16–23.

[Pit97] J. Pitkow, *In Search of Reliable Usage Data on the WWW*, Proceedings of the Sixth International WWW Conference (Santa-Clara, CA), 1997, p. 451–463.

[PP99] J. Pitkow et P. Pirolli, *Mining Longest Repeating Subsequences to Predict World Wide Web Surfing*, Proceedings of USITS'99: The 2nd USENIX Symposium on Internet Technologies & Systems, 1999.

[PPPS01] D. Pierrakos, G. Paliouras, C. Papatheodouro, et C. D. Spyropoulos, *KOINOTITES: A Web Usage Mining Tool for Personalization*, Proceedings of the Panhellenic Conference on Human Computer Interaction, PC-HCI (Patras), 2001, p. 231–236.

[PPPS03] D. Pierrakos, G. Paliouras, C. Papatheodorou, et C. D. Spyropoulos, *Web Usage Mining As a Tool for Personalization: A Survey*, User Modeling and User-Adapted Interaction **13** (2003), no. 4, 311–372.

[PPR96] P. Pirolli, J. Pitkow, et R. Rao, *Silk From a Sow's Ear: Extracting Usable Structures From the Web*, Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI'96 (Vancouver), ACM Press, 1996.

[Qui93] J. R. Quinlan, *C4.5: Programs for machine learning*, M. Kaufmann, 1993.

[Reb67] A. S. Reber, *Implicit Learning of Artificial Grammars*, Journal of Verbal Learning and Verbal Behaviour **6** (1967), 855–863.

[RST95] D. Ron, Y. Singer, et N. Tishby, *the Learnability and Usage of Acyclic Probabilistic Finite Automata*, Proc. of the Eighth Annual ACM Workshop on Computational Learning Theory, 1995, p. 31–40.

[SBKF01] C. Shahabi, F. Banaei-Kashani, et J. Faruque, *A Reliable, Efficient, and Scalable System for Web Usage Data Acquisition*, WEBKDD Workshop: Mining Log Data Across All Customer TouchPoints, 2001.

[SM06] W. Shi et Y. Mao, *Performance Evaluation of Peer-to-peer Web Caching Systems*, Journal of Systems and Software **79** (2006), no. 5, 714–726.

[TD00] F. Thollard et P. Dupont, *Inference Grammaticale Probabiliste Utilisant La Divergence De Kullback-Leibler Et Un Principe De Minimalité*, Conférence D'APprentissage, 2000, p. 259–275.

[TDH00] F. Thollard, P. Dupont, et C. de la Higuera, *Probabilistic DFA Inference using Kullback-Leibler Divergence and Minimality*, Proceeding of ICML'00, Morgan Kaufmann, San Francisco, CA, 2000, p. 975–982.

[Tho00] F. Thollard, *Inférence grammaticale probabiliste pour l'apprentissage de la syntaxe en traitement de la langue naturelle*, Ph.D. thesis, Université Jean Monnet, Saint-Étienne, France, Juillet 2000.

[Tur50] A. M. Turing, *Computing Machinery and Intelligence*, MIND **59** (1950), 433–460.

[Val84] L. G. Valiant, *A Theory of the Learnable*, Communications of the Association for Computing Machinery **27** (1984), no. 11, 1134–1142.

[VG] Inc. Volatile Graphix, *IP Addresses of Search Engine Spiders*, http://www.iplists.com/.

[VS94] 1.0 edition V. Shea, *Netiquette*, Albion Books, May 1994, ISBN : 0-9637025-1-3.

[W3C92] W3C, *Html*, http://www.w3.org/History/19921103-hypertext/hypertext/-WWW/MarkUp/MarkUp.htm/, 1992.

[W3C98] ———, *Extended markup language (xml) 1.0*, http://www.w3.org/TR/-1998/REC-xml-19980210, 1998.

[W3C99a] ———, *Hypertext markup language – html 4.01*, http://www.w3.org/TR/-html401/, 1999.

[W3C99b] ———, *Hypertext transfer protocol – http/1.1*, http://www.w3.org/-Protocols/rfc2616/rfc2616.txt, jun 1999.

[W3C99c] ———, *Web characterization terminology & definitions sheet*, http://www.w3.org/1999/05/WCA-terms/, 1999.

[W3C00] ———, *Xhtml 1.0 : The extensible hypertext markup language*, http://www.w3.org/TR/2000/REC-xhtml1-20000126, 2000.

[W3C04] ———, *Document object model (dom) level 3 core specification*, http://www.w3.org/TR/DOM-Level-3-Core/, 2004.

[ÉF78] État Français, *Loi relative à l'informatique, aux fichiers et aux libertés*, Journal Officiel de la République Française, jan 1978, Loi n°78-17 du 6 janvier 1978.

[ÉF92] ———, *Code pénal*, LOI n°92-686 du 22 juillet 1992, jui 1992.

Table des figures

Liste des tableaux

Liste des Algorithmes

Résumé

Les travaux présentés se situent dans le cadre d'extraction de connaissance à partir de données. Un contexte d'étude intéressant et d'actualité a été choisi : les sites web adaptatifs. Pour mettre en œuvre, de manière la plus automatique possible, de tels sites adaptés aux utilisateurs, nous décidons d'apprendre des modèles d'utilisateurs ou, plus précisément, de leurs types de navigations sur un site web donné. Ces modèles sont appris par inférence grammaticale.

Les données disponibles liées au contexte du Web sont particulièrement difficiles à récupérer proprement. Nous choisissons de nous focaliser sur les fichiers de *logs* serveur en supprimant le bruit inhérent à ces derniers.

L'inférence grammaticale peut généraliser ses données d'entrée pour obtenir de bons modèles de langages. Nous travaillons sur les mesures de similarité entre langages pour l'évaluation de la qualité des modèles appris. L'introduction d'une mesure euclidienne entre modèles de langages représentés sous forme d'automates permet de pallier les problèmes des métriques existantes. Des résultats théoriques montrent que cette mesure a les propriétés d'une vraie distance.

Enfin, nous présentons divers résultats d'expérimentation sur des données du web que nous pré-traitons avant d'apprendre grâce à elles des modèles utilisateurs issus de l'inférence grammaticale stochastique. Les résultats obtenus sont sensiblement meilleurs que ceux présents dans l'état de l'art, notamment sur les tâches de prédiction de nouvelle page dans une navigation utilisateur.

Abstract

Our work is about Knowledge Discovery and Data Mining. We focus on web data including server log files. In order to know automatically how to adapt a web site, we decide to learn grammatical models about users behaviors.

We show in this work how the web data are difficult to acquire in order to use them in a grammatical inference process. We try to eliminate the almost totality of the noise which is present in these data.

We also show how grammatical inference can learn good models by generalizing enough its input data. We explain how difficult the evaluation of the quality of learned models is, and we introduce an euclidean measure between languages models represented by automata. We prove that this measure is a true distance in a mathematical sense.

Finally, we propose our experimentation results: we show that our method (from the prepossessing of the data to the evaluation of learned models) gives better success rates for the new page prediction task which is very common in web usage mining.